

目 录

第 0 章 关于本书与随书光盘	1
0-1 本书简介	2
0-2 本书结构	2
0-3 本书范例	4
0-4 软件和版本	4
0-5 本书练习题	4
0-6 本书光盘内容	4
第 1 章 Swing 简介	5
1-1 Swing 的概观	6
1-2 Swing 结构	6
1-3 Swing 常用的 package	8
1-4 Swing 组件	8
1-5 本章总结	16
1-6 本章习题	16
第 2 章 Swing 的基本概念与使用	17
2-1 Java 窗口的演进	18
2-1-1 什么是 Swing, 什么又是 lightweight component	18
2-1-2 您使用的 JDK 有包含 Swing 吗	19
2-2 如何编译与运行包含 Swing 程序代码的 JAVA 程序	19
2-2-1 下载 JDK	19
2-2-2 安装 JDK	21
2-2-3 设置 JDK 的操作环境	23
2-3 编写第一个 Java 程序	27
2-3-1 编译与运行 Java Application	28
2-3-2 编译与运行 Java Applet	30
2-4 本章总结	33
2-5 本章习题	33
第 3 章 使用版面管理器 (Layout Managers)	35
3-1 Swing 的版面结构	36
3-2 版面管理器 (Layout Manager)	39
3-2-1 Layout Manager 的种类与介绍	39
3-2-2 BorderLayout 的使用	39

3-2-3	FlowLayout 的使用	41
3-2-4	GridLayout 的使用	44
3-2-5	CardLayout 的使用	46
3-2-6	GridBagLayout 的使用	49
3-2-7	BoxLayout 的使用	54
3-2-8	不使用版面管理器	62
3-3	本章总结	63
3-4	本章习题	64
第 4 章	事件处理 (Event Handling)	65
4-1	事件处理	66
4-2	事件处理范例说明	71
4-2-1	ActionEvent、WindowEvent 与事件处理的多种写法	71
4-2-2	相同组件事件的处理	77
4-2-3	鼠标事件处理	80
4-2-4	键盘事件处理	87
4-3	本章总结	89
4-4	本章习题	90
第 5 章	窗口与面版 (Frame、Pane 与 Panel) 的使用与介绍	91
5-1	JFrame 的使用	92
5-2	Swing 的容器结构与 JLayeredPane 的使用	94
5-3	JInternalFrame 的使用	101
5-4	JPanel 的使用	104
5-5	JSplitPane 的使用	106
5-6	JTabbedPane 的使用	108
5-7	JScrollPane 的使用	114
5-8	JScrollBar 的使用	119
5-9	本章总结	121
5-10	本章习题	122
第 6 章	标签与按钮的使用与介绍	123
6-1	Border 的使用	124
6-2	Icon 的使用	131
6-3	JLabel 的使用	135
6-4	JButton 的使用	139
6-4-1	在 JButton 上使用 Rollover 图像变化	141
6-4-2	在 JButton 上设置快捷键	143
6-4-3	设置默认按钮	145
6-5	JToggleButton 的使用	147
6-6	本章总结	151

6-7 本章习题	151
第 7 章 复选框、选项按钮、列表方框、下拉式列表的使用与介绍	153
7-1 使用 JCheckBox 组件	154
7-1-1 构造 JCheckBox 组件	154
7-1-2 JCheckBox 事件处理	156
7-2 JRadioButton 的使用	159
7-2-1 构造 JRadioButton 组件与事件处理	160
7-3 JList 的使用	164
7-3-1 建立一般的 JList	164
7-3-2 利用 ListModel 构造 JList	168
7-3-3 建立有图像的 JList	174
7-3-4 JList 的事件处理	177
7-4 JComboBox 的使用	181
7-4-1 建立一般的 JComboBox	182
7-4-2 利用 ComboModel 构造 JComboBox	184
7-4-3 建立有图像的 JComboBox	188
7-4-4 建立可自行输入的 JComboBox	195
7-4-5 JComboBox 的事件处理	196
7-5 本章总结	200
7-6 本章习题	200
第 8 章 表格 (Table) 的使用与介绍	201
8-1 使用 JTable 组件	202
8-2 TableModel	206
8-3 AbstractTableModel	208
8-4 TableColumnModel	214
8-5 SelectionModel	217
8-6 DefaultTableModel	220
8-7 JTable 的事件处理	224
8-8 本章总结	228
8-9 本章习题	228
第 9 章 文字输入组件的使用与介绍	229
9-1 认识 Swing 的文字输入组件	230
9-2 使用 JTextField 组件	231
9-2-1 构造一般的 JTextField 组件	231
9-2-2 利用 Document 构造 JTextField	235
9-2-3 JTextField 的事件处理	237
9-3 使用 JPasswordField 组件	238

9-3-1	构造一般的 JPasswordField 组件	239
9-3-2	利用 Document 构造 JPasswordField	242
9-3-3	JPasswordField 的事件处理	244
9-4	使用 JTextArea 组件	245
9-4-1	构造的 JTextArea 组件	246
9-4-2	JTextArea 的事件处理	252
9-5	使用 JEditorPane 组件	257
9-5-1	构造 JEditorPane 组件	258
9-5-2	JEditorPane 的事件处理	263
9-6	使用 JTextPane 组件	265
9-6-1	JTextPane 的特性	266
9-6-2	构造 JTextPane 组件	266
9-7	本章总结	269
9-8	本章习题	270
第 10 章	树 (Tree) 的使用与介绍	271
10-1	使用 JTree 组件	272
10-2	以 Hashtable 构造 JTree	274
10-3	以 TreeNode 构造 JTree	276
10-4	以 TreeModel 构造 JTree	278
10-5	改变 JTree 的外观	281
10-6	更换 JTree 节点图案	283
10-7	JTree 的事件处理模式	286
10-7-1	处理 TreeModeEvent 事件	286
10-7-2	处理 TreeSelectionEvent 事件	295
10-8	JTree 的其他操作	298
10-9	本章总结	298
10-10	本章习题	299
第 11 章	对话框 (Option Pane 与 Dialog) 的使用与介绍	301
11-1	使用 JDialog 组件	302
11-1-1	在 JFrame 上建立 JDialog	303
11-1-2	在 JApplet 上建立 JDialog	307
11-2	使用 JOptionPane 类的静态方法	310
11-2-1	输出 Message Dialog	313
11-2-2	输出 Confirm Dialog	316
11-2-3	输出 Input Dialog	319
11-2-4	输出 Option Dialog	321
11-2-5	输出 Internal Dialog	323
11-3	使用 JOptionPane 组件建立对话框	326

11-4	本章总结	328
11-5	本章习题	328
第 12 章	菜单与工具栏的使用与介绍	329
12-1	使用 JMenuBar 组件	330
12-2	使用 JMenu 组件	330
12-2-1	构造 JMenu 组件	331
12-3	使用 JMenuItem 组件	332
12-3-1	构造 JMenuItem 组件	333
12-3-2	JMenuItem 的事件处理	342
12-4	使用 JCheckBoxMenuItem	345
12-4-1	构造 JCheckBoxMenuItem 组件	345
12-5	使用 JRadioButtonMenuItem 组件	349
12-5-1	构造 JRadioButtonMenuItem 组件	349
12-6	使用 JToolBar 组件	353
12-6-1	构造 JToolBar 组件	354
12-6-2	在 JToolBar 组件中加入 ToolTip	361
12-7	使用 JPopupMenu 组件	363
12-7-1	构造 JPopupMenu 组件	364
12-8	本章总结	370
12-9	本章习题	370
第 13 章	文件选择对话框、颜色选择对话框、分隔线的使用与介绍	371
13-1	使用 JFileChooser 组件	372
13-1-1	建立一个简单的 JFileChooser 对话框	372
13-1-2	建立可选择文件类型的 JFileChooser 对话框	377
13-1-3	建立具有特殊文件类型图标 JFileChooser	381
13-2	建立颜色选择对话框 (JColorChooer)	386
13-2-1	轻松输出颜色选择对话框	386
13-2-2	建立 JColorChooser 对象输出颜色选择对话框	390
13-2-3	将 JColorChooser 置于一般容器中显示	393
13-2-4	改变 JColorChooser 的颜色选择面板	395
13-3	建立分隔线 (JSeparator)	395
13-4	本章总结	397
13-5	本章习题	398
第 14 章	滑动杆 (Slider)、时间控制 (Timer)、进度组件 (Progress) 的使用与介绍	399
14-1	使用 JSlider 组件	400
14-1-1	建立 JSlider 组件	400

14-1-2	为 JSlider 组件加入刻度	404
14-1-3	自定义 JSlider 标记名称	407
14-2	使用 Timer 组件	409
14-3	使用 Progress Bar 组件	413
14-4	使用 Progress Monitor 组件	417
14-5	使用 Progress Monitor Input Stream 组件	421
14-6	本章总结	429
14-7	本章习题	429
第 15 章	创造用户最熟悉的环境 (Look and Feel)	431
15-1	为什么要用 Look and Feel	432
15-2	什么是 Look and Feel	432
15-3	在 Java 中如何使用 Look and Feel	434
15-3-1	Look and Feel 范例一	434
15-3-2	Look and Feel 范例二	439
15-4	本章总结	448
15-5	本章习题	448
第 16 章	整合范例	449
16-1	建立窗口	450
16-2	菜单与工具栏 (Menus and Toolbars)	451
16-2-1	菜单 (Menus)	451
16-2-2	工具栏 (Toolbars)	458
16-3	各种常用的互动组件 (ComboBox、CheckBox and Radio)	460
16-4	其他常用的组件 (Slider、Tree、Table and Password Field)	466
16-4-1	Slider	469
16-4-2	Tree Structure	470
16-4-3	Table Structure	472
16-4-4	Password Field	473
16-4-5	Progress Bar	475
16-5	其他常用的组件 2 (JOptionPane、JEditorPane)	476
16-6	整合范例	482
16-6-1	MainFrame 程序初始化	483
16-6-2	建立 Desktop Pane	483
16-6-3	建立菜单	484
16-6-4	建立工具栏	486
16-6-5	建立快捷菜单 (Popup Menu)	488
16-6-6	处理编辑器中的排列方式	488
16-6-7	处理新增文件	489
16-6-8	处理读取文件、关闭文件、离开程序	492

16-6-9	处理 Undo 与 Redo.....	495
16-6-10	处理 Copy、Cut、Paste 操作	498
16-6-11	处理改变粗斜体、下划线、颜色、字体变换与字号操作	499
16-6-12	处理插入图片与组件事件	502
16-6-13	实现 Demo 菜单中的各种功能.....	504
16-6-14	实现 Help 菜单中的各种功能.....	505
16-7	编辑器运行结果	505
16-8	MainFrame 程序内容.....	508
16-9	一些小技巧	519
16-10	本章总结	521
16-11	本章习题	521

0

关于本书与随书光盘

深入浅出

0-1 本书简介

本书主要针对 Java 窗口程序设计来做深入探讨,对于一般的 Java 语法与面向对象概念我们在此将不多做讨论,原因是 Swing 本身的内容已经相当丰富,我们希望读者在阅读完此书后能够充分发挥 Swing 的功能,设计出相当实用与出色的用户界面,而非仅是对 Swing 只有粗浅的认识。

JDK1.2 以后,已将 Swing 包含其中,提供 Java 强大的窗口界面功能。在本书中我们将详细地介绍 Swing 所提供的功能,对于已熟悉 Java 基本语法的读者而言,此书将可帮助您快速发展程序界面,是一本相当实用的工具书。然而窗口程序设计本身是一门学问,也是一种艺术,本书仅提供实作界面的各种方法与技术,至于有关设计风格与界面流畅性,则有赖您的经验、想象力与配合其他相关书籍了。

0-2 本书结构

本书共有 16 章,大致上可分成三个部分,第一部分为 1~4 章,介绍 Swing 的基本概念,包括 Swing 的由来、Swing 的结构、版面的管理、事件的处理等等。第二部分为 5~14 章,讲述 Swing 的各个组件,包括组件的使用、外观的变化、可能产生的事件等等。第三部分为 15~16 章,解说 Swing 的应用,为以前所介绍的组件做一个整理,说明了 Swing Look and Feel 的功能,并在最后以一个综合性例子贯穿全文,读者可从此例子了解到以 Java 开发各种应用软件比起其他程序语言要来得简单、快速、而且精简多了。若读者已具有 Swing 的基本概念,了解事件处理模式与版面管理,可跳过 1~4 章直接阅读 5~14 章,此部分可依所需跳章阅读。下面我们列出本书各章的简要说明。

第 0 章:关于本书与随书光盘

说明本书结构与其他内容。

第 1 章:Swing 简介

说明 Swing 优于 AWT 的种种原因,以及 Swing 当初设计的理念,设计的方法与结构。

第 2 章:Swing 的基本概念与使用

说明如何顺利地运行含有 Swing 组件的 Java 程序,如何取得最新的 JDK 或 JRE 等等。

第 3 章:使用版面管理器

说明各种版面管理器的应用,这是摆放 Swing 组件的第一步。

第 4 章:事件处理

事件处理在用户界面上占有极重要的地位,读者应该好好熟悉此章所提到的各种事件处理技巧。



第 5 章：窗口与面版的使用与介绍

说明 JFrame、JPanel、JLayeredPane、JSplitPane、JTabbedPane、JScrollPane 与 JScrollBar 的使用与介绍。JFrame 与 JPanel 是我们最常用的容器，读者一定要对它们相当熟悉。

第 6 章：标签和按钮的使用与介绍

此章一开始我们先介绍 Icon 与 Border 的使用方法，这两个组件常在 JButton 与 JLabel 中出现。接着我们详细说明 JButton、JLabel 与 JToggleButton 的运用。

第 7 章：复选框、选项按钮、列表方框、下拉式列表的使用与介绍

此章介绍 JCheckBox、JRadioButton、JList 与 JComboBox 的使用方法，并说明它们之间的差异。

第 8 章：表格的使用与介绍

此章介绍 JTable 的使用方法，读者在阅读完本章后将可轻松地写出一个电子表格程序。

第 9 章：文字输入组件的使用与介绍

此章介绍 JTextField、JPasswordField、JTextArea、JEditorPane 与 JTextPane 的使用方法，这个部分可说是 Swing 中最复杂、最难的地方，但在本章中我们先稍微简单地介绍各组件的功能，较复杂的应用，读者可在第 16 章中看到。

第 10 章：树的使用与介绍

此章介绍 JTree 的使用方法，读者可利用此章内容作出文件管理器的功能出来。

第 11 章：对话框的使用与介绍

此章介绍 JDialog 与 JOptionPane 的使用方法，并说明了它们之间的差异与使用。

第 12 章：菜单和工具栏的使用与介绍

此章介绍 JMenuBar、JMenu、JMenuItem、JCheckBoxMenuItem、JRadioButtonMenuItem、JToolTip 与 JPopupMenu 的使用方法，这些组件我们在一般软件中时常看到，也非常的实用，更令人兴奋的是以 Java 来设计这些组件，将是一件相当轻松的工作。

第 13 章：文件选择对话框、色彩选择对话框、分隔线的使用与介绍

此章介绍 JFileChooser、JColorChooser 与 JSeparator 的使用方法。读者将对文件与颜色的选择有一个新的认识。

第 14 章：滑动杆、时间控制、进度组件的使用与介绍

此章介绍 JSlider、Timer、JProgressBar、ProgressMonitor、ProgressMonitorInputStream 的使用方法。若您想控制时间进度或是设计定时器，您可不能错过此章的内容。

第 15 章：创造用户最熟悉的环境

此章介绍什么是 Look and Feel、为什么要使用 Look and Feel 以及如何使用 Look and Feel。

第 16 章：整合范例

以文本编辑器例子作为复习各个组件的综合性范例。此章我们将一步一步带领读者完成一个较具规模的程序范例。

0-3 本书范例

为了让读者轻松地了解 Swing 各个组件的特色，我们在本书中编写了相当多的范例供读者参考。

本书范例中的序号都是为了说明方便起见所设定的，所有的实际文件内容都不应该含有这些序号。



注意

0-4 软件和版本

本书所采用的 Java 版本是 JDK1.4，运行环境为 JRE1.4 版，读者可在 Sun 公司或各大 FTP 网站中找到此版本或更新版本的 Java 环境。

0-5 本书练习题

本书各章（不含第 0 章）后面均含有练习题，作为读者学习完该章后复习之用。

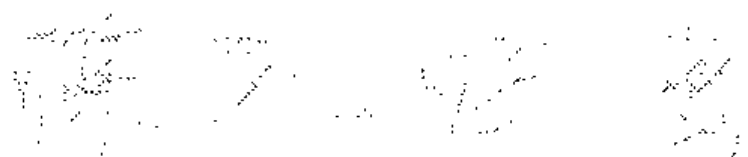
0-6 本书光盘内容

在光盘的“\exam”目录中含本书所有的程序范例。

1

Swing 简介

Swing 是目前 Java 不可或缺的窗口工具组，在 Swing 尚未推出之前，要撰写一个 Java 窗口程序必须使用 AWT (Abstract Window Toolkit) 的套件，AWT 最大的缺点在于使用上相当没有灵活性且缺乏效率。在本章中，我们将说明 Swing 为何优于 AWT 的种种原因，并对 Swing 的 MVC 结构有一个新的认识。您可在以后的章节慢慢体会出 Swing 组件设计的理念，让我们一起遨游 Swing 的世界吧！



1-1 Swing 的概观

Swing 是目前 Java 不可或缺的窗口工具组，在 Swing 尚未推出之前，要编写一个 Java 窗口程序必须使用 AWT (Abstract Window Toolkit) 的包，AWT 最大的缺点在于使用上相当没有弹性，例如您无法任意地改变组件的外观，甚至想在一个按钮上加上图案都是非常困难的。这个原因主要是 AWT 中大部分的组件均含有 native code，这样的做法相当容易理解，因为窗口中的每个组件都可能与操作系统相互沟通（不管是输入或输出），而我们使用的操作系统并不是由 Java 程序所写成，因此要与操作系统相互沟通必须使用与操作系统兼容的程序语言，产生所谓的对等 (Peer) 组件，这样的做法所需付出的代价除了刚刚提到的没有弹性外，也可能耗费大量的系统资源，因为建立一个 AWT 组件就等于建立一个对等对象，然后由此对等对象直接跟操作系统沟通，因此若我们想建立一个 10×10 大小的电子表格，至少就必须产生 101 个对等对象（可能是由 100 个 TextField 组件加上一个 Frame 组件组成），这样的做法似乎太没有效率了。

1998 年 Sun 推出含有 Swing 配件的 JDK1.2 版本后，彻底解决了上述的问题。Swing 并不是用来取代原有的 AWT package，事实上当您使用 Swing 组件时常常还是需要使用旧有的 AWT 功能，例如事件的处理 (Event Handle) 或是版面的配置 (Layout Manager)，因此您可以把 Swing 与 AWT 看成是相辅相成的两大工具组。不过 Swing 组件大部分均是由纯 Java 程序所写成的（只有 JFrame、JDialog、JWindow 与 JApplet 不是，我们将在第 3 章提到），这样的做法可以让我们很随意地改变组件的外观，甚至是动态地改变 (Runtime 时改变外观)，如动态地更改 Look and Feel 功能（此部分将在第 15 章中介绍）。此外，由于大部分的 Swing 组件是由纯 Java 所写成的，因此产生的 Swing 组件就不会产生相对应的对等组件，可减少系统资源的使用，也可以增加系统的稳定度，而所有与操作系统间的交互将通过最上层组件 (JFrame、JDialog、JWindow 与 JApplet) 来完成。

Swing 中只有 JFrame、JDialog、JWindow 与 JApplet 不是纯 Java 所写成，主要是窗口画面总要有跟操作系统沟通的渠道，这样才知道用户是不是敲了键盘、按了鼠标或关闭了窗口。当您在 JFrame 上绘制了 100 个 JTextField 组件时，实际上只会产生一个对等组件，也就是由 JFrame 所产生，利用这个唯一的对等组件来跟操作系统相互沟通，这样就能节省相当多的系统资源。再者，不同的操作系统所对应的对等组件可能有不同的效果，减少对等物的产生将有利于系统的稳定度。



注意

1-2 Swing 结构

Swing 原先是利用 MVC (Model-View-Controller) 的概念衍生而出，这个模式最早是应用在 SmallTalk 语言上。MVC 的概念如下：

Model: 存储组件数据的地方，如 JCheckBox 中的 Model 将存储所有选项的值，并以 Boolean 值来表示，而 JTable 的 Model 则会存储所有表格上的数据。

View: 显示组件的外观。

Controller: 处理用户在组件上的操作，并将改变后的数据存储存储在 Model 中。

整个 MVC 的图形表示如图 1-1 所示。

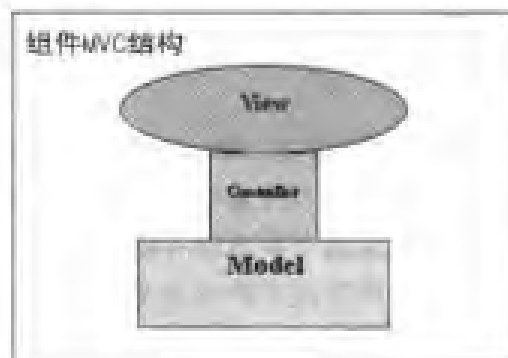


图 1-1

运用 MVC 结构可以使程序更具有对象化的特性，也更容易维护。例如您可以将同一份数据用不同的外观显示出来；您可以把公司的财务营收数据利用表格显示，也可以利用长条图显示，但所用的均是同一份数据。由于 MVC 具有清楚的模块结构，因此 Swing 在设计上也遵循这样的概念，但做了一点小小的改变。原先独立出来的 View 与 Controller 在 Swing 中是将它合并在一起的，并称为委托式 UI 界面（UI-delegate），原因是 Swing 的发展小组认为这两者具有紧密的相关性，若分开设计并没有太多的好处，且会提高设计上的困难度，所以原先的 MVC 结构就被更改成 M-UI 结构，如图 1-2 所示。

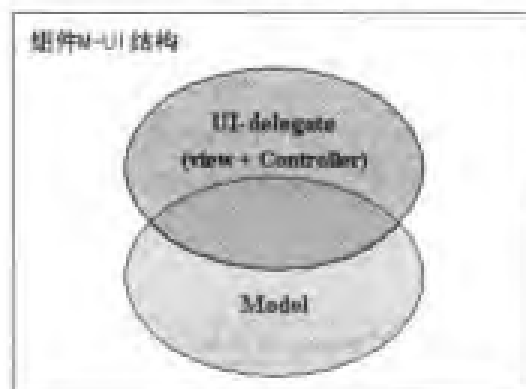


图 1-2

不管 Swing 的 M-UI 结构如何，只要读者有 MVC 的概念就不难想象 Swing 组件设计的原理。在 Swing 的 MVC 概念中，Model 的概念最为重要与复杂。Swing 组件中的 Model 可分成 3 个种类，分述如下：

第一种：存储组件内将被用来操作的数据。此种模式最常被用来显示不同的数据显示方式，如 TableModel、ListModel、ComboBoxModel、Document 等。TableModel 是存储 JTable 上所有字段的数据，ListModel 是存储 JList 上所有项目的数据，Document 是存储文字输入组件（JTextField、JTextArea 等）内的所有内容，我们将在以后章节介绍这些模式。适时地操作这些模式将有助于程序的简化，例如同一份员工绩效数据，老板看到的内容跟员工看到的内容就应该有所不同，

若不使用上面这些模式来配合，您可能必须编写两个相似的程序，这样的程序看起来就不精简。

第二种：存储控制组件操作模式的数据。例如在 `JList` 或 `JTable` 中，我们可以使用 `ListSelectionModel` 来决定用户一次是否能选择多列的数据，或是 `JTable` 的 `TableColumnModel` 可用来设置 `JTable` 中每个格子的组件类型（如 `JTextField` 或 `JComboBox` 类型）。

第三种：存储组件本身性质的数据模式。例如 `JSlider` 或 `JProgressBar` 组件都有 `minimum`、`maximum`、`extent` 与 `value` 四个参数值，用来代表组件的最小值、最大值、延伸区值与初始设置值，这四个参数值存储在 `BoundedRangeModel` 中。

当用户与界面有任何交互时，所有改变的值会更改至 `Model` 中原有设置的数据，此时 `Controller` 会负责调用更改画面与变更 `Model` 数据的操作。

1-3 Swing 常用的 package

由于 `Swing` 功能相当的强大与复杂，因此与 `Swing` 组件有关的类依功能分散在不同的 `package` 中，如表 1-1 所示：


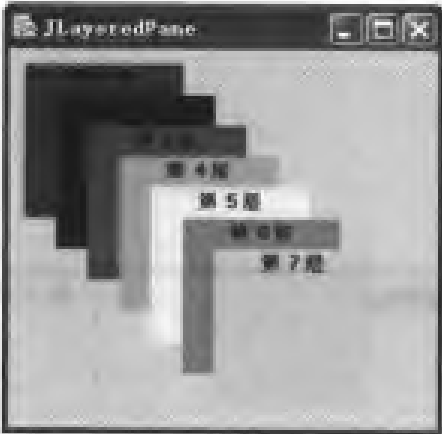
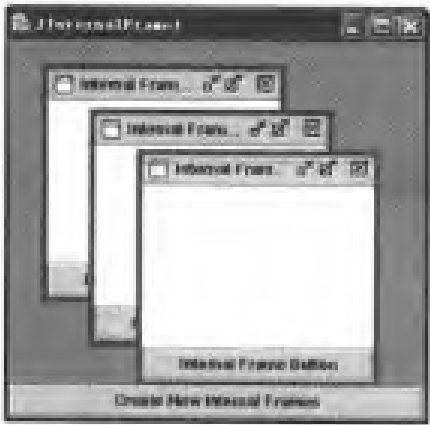


表 1-1

package	内容
<code>javax.swing</code>	最常使用的 <code>package</code> ，里面包含了各种 <code>Swing</code> 组件的类
<code>javax.swing.border</code>	包含与 <code>Swing</code> 组件外框有关的类
<code>javax.swing.colorchooser</code>	针对 <code>Swing</code> 调色盘组件（ <code>JColorChooser</code> ）所设计的类
<code>javax.swing.event</code>	处理由 <code>Swing</code> 组件产生的事件，有别于 <code>AWT</code> 事件
<code>javax.swing.filechooser</code>	包含针对 <code>Swing</code> 文件选择对话框（ <code>JFileChooser</code> ）所设计的类
<code>javax.swing.plaf</code> <code>javax.swing.plaf.basic</code> <code>javax.swing.plaf.metal</code> <code>javax.swing.plaf.multi</code>	处理 <code>Swing</code> 组件外观相关的类
<code>javax.swing.table</code>	针对 <code>Swing</code> 表格组件（ <code>JTable</code> ）所设计的类
<code>javax.swing.text</code> <code>javax.swing.text.html</code> <code>javax.swing.text.html.parser</code> <code>javax.swing.text.rtf</code>	包含与 <code>Swing</code> 文字组件相关的类
<code>javax.swing.tree</code>	针对 <code>Swing</code> 树状组件（ <code>JTree</code> ）所设计的类
<code>javax.swing.undo</code>	提供 <code>Swing</code> 文字组件 <code>Redo</code> 或 <code>Undo</code> 的功能

1-4 Swing 组件

本书将介绍所有可能用到的 `Swing` 组件，并深入浅出地为读者详细介绍每个组件的使用方式与相关功能。我们也为每个组件举出了各种实例，让读者对 `Swing` 组件的印象更为深刻。以下是本书将介绍的 `Swing` 组件列表，如表 1-2 所示。


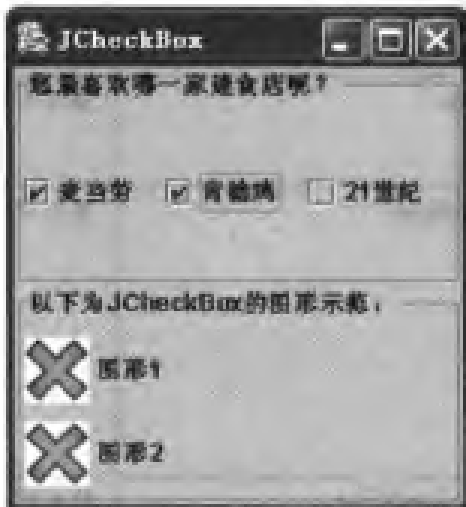
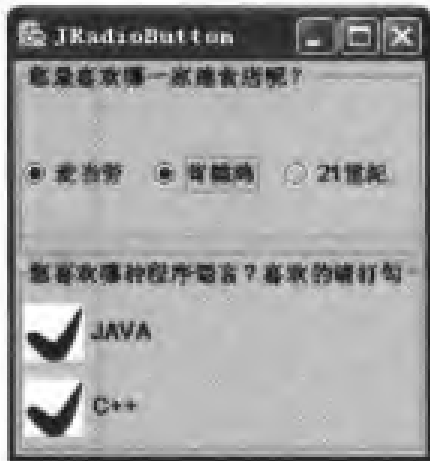

表 1-2

Swing 组件	相关章节	图 标
JFrame	第 5 章	 A window titled 'JFrameDemo' with a button labeled 'Click me to get new Window'.
JLayeredPane	第 5 章	 A window titled 'JLayeredPane' showing three overlapping rectangular regions labeled '第 4 层' (Layer 4), '第 5 层' (Layer 5), and '第 7 层' (Layer 7).
JInternalFrame	第 5 章	 A window titled 'JInternalFrameDemo' showing three overlapping internal frames, each titled 'Internal Frame...', and a button labeled 'Internal Frame Button' at the bottom.
JPanel	第 5 章	 A window titled 'JPanelDemo' containing five labels: 'Label 1' at the top, and 'Label 2', 'Label 3', 'Label 4', and 'Label 5' arranged in a grid below.
JSplitPane	第 5 章	 A window titled 'JSplitPaneDemo' showing a split pane with 'Label 1' and 'Label 2' in the top section, and 'Label 3' in the bottom section.

续上表

Swing 组件	相关章节	图 标
JTabbedPane	第 5 章	
JScrollPane	第 5 章	
JScrollBar	第 5 章	
JLabel	第 6 章	
JButton	第 6 章	




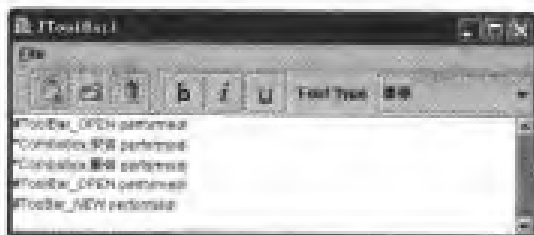

线上表

Swing 组件	相关章节	图 标
JToggleButton	第 6 章	
JCheckBox	第 7 章	
JRadioButton	第 7 章	
JList	第 7 章	


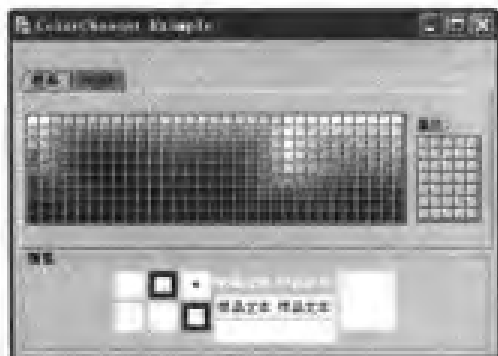



续上表

Swing 组件	相关章节	图 标
JTextPane	第 9 章	
JTree	第 10 章	
JDialog	第 11 章	
JOptionPane	第 11 章	
JMenu	第 12 章	

续上表

Swing 组件	相关章节	图 标
JMenuItem	第 12 章	
JCheckBoxMenuItem	第 12 章	
JRadioButtonMenuItem	第 12 章	
JToolBar	第 12 章	
JPopupMenu	第 12 章	

续上表

Swing 组件	相关章节	图 标
JFileChooser	第 13 章	
JColorChooser	第 13 章	
JSeparator	第 13 章	
JSlider	第 14 章	
JProgressBar	第 14 章	

1-5 本章总结

Swing 是 Java 窗口程序不可或缺的包，有别于以往 AWT 包没有弹性、缺乏效率的缺点，Swing 可以提供更丰富的视觉感受。在本章中，我们说明了 Swing 如何优于 AWT 的种种原因，并对 Swing 的 MVC 结构有一个新的认识。您可以在以后的章节慢慢体会出 Swing 组件设计的理念，让我们一起遨游 Swing 的世界吧！

1-6 本章习题

1. 试说明 Swing 组件设计的理念优于 AWT 的原因。
2. 试说明 Swing 的结构概念。
3. 试解释为什么 Swing 在设计上将 View 与 Controller 合在一起，这样的做法有什么好处？

2

Swing 的基本概念与使用

在此章中我们将介绍什么是 Swing、Swing 的历史、Swing 包含了哪些组件、Swing 与 AWT 有何差异等等。当然，为了让读者顺利地运行 Swing 程序，本章详细地说明了运行 Swing 程序时所需注意的项目：包括 Java 插件的设置与如何改写 HTML 文件。让 IE 浏览器也能正确运行 Swing 程序。

深入淺出

2-1 Java 窗口的演进

以前 Java 尚未推出 Swing 之前,要设计一个 Java 窗口程序,必须要通过 AWT (Abstract Window Toolkit) 组件。AWT 组件虽然提供了一些构造窗口的基本需求,如文字字段 (TextField)、复选框 (Check Box)、按钮 (Button)、对话框 (Dialog)、菜单 (Menu) 等等,但在使用上非常不方便且呆板。例如:我们无法改变按钮的形状,也无法在按钮上加上图标;对于版面管理 (Layout manager) 的限制相当多,而对图形处理的支持也很少……,相信这些均是以前 Java 程序设计师的一大麻烦。如今, Sun 公司已经把 Swing 列为 Java 的基本包之一,过去的种种麻烦已不再困扰着 Java 程序设计师。今天要设计一个漂亮的窗口界面,例如列表 (Table)、树状图 (Tree)、小图标 (Icon) 等,均可以很容易地做出来,且更具有弹性。这些当然要归功于 Java 开发研究者的功劳啰!

AWT 之所以不容易改动与变化的主要原因,是由于 AWT 的许多组件均含有 C 语言的成分,即利用 native code 的方式来撰写。当初会这样设计,是因为利用 C 语言来设计窗口组件,相当的容易与快速,也方便与操作系统相沟通。而且当初 Java 并不像现在红透半边天,因此为了快速设计出 Java 整个雏形,才采用这样的方式设计 AWT。也由于这些组件具有许多 C 语言的成分,因此用户便无法自行去改动它的形状,也无法任意做出其他的变化。



注意

2-1-1 什么是 Swing, 什么又是 lightweight component

要了解什么是 Swing,我们必须先了解 JFC 这个缩写字。JFC 是 Java™ Foundation Classes 的缩写,其功能是提供程序设计者设计图形用户界面 (GUI) 用,在 1997 年时,由 JavaOne developer conference 所提出。其主要包含 5 个部分:

1. AWT 组件: 旧有的窗口组件包。
2. Swing 组件: 新的窗口组件包,即本书将介绍的重点
3. Accessibility API: 提供一种更先进的沟通界面。例如: 语音输入或触摸式屏幕,可帮助一些不方便使用键盘的人来操作计算机。当然,远程语音遥控或触摸式计算机,同样也可给一般人的生活带来极大方便。
4. Java 2D API: 提供更强大的图形处理函数。
5. 支持 Drag and Drop 功能: 您应该有在 Windows 上开两个窗口做文件的 Drag and Drop (Copy and Paste) 的经验吧! 现在 Java 也提供了这样类似的功能,让您在两个相同的 Java 界面,甚至是 Java 与其他应用程序界面作数据交换的操作。

在 Java1.1 时,Swing 被包含在外挂组件 JFC1.1 中,用户要使用 Swing 组件必须自行加装这个外挂组件,并不是十分方便。到了 Java 2 以后,Swing 便成为 Java 的标准包之一,用户安装完 Java 后就可以直接使用 Swing 了。也许是 Swing 的组件占了 JFC 绝大多数,且最常被大家所使用,因此有些人会误称 JFC 就是 Swing,其实这是不正确的说法,因为 Swing 只是 JFC 所包含的其中一个部分。

我们常会听到许多人说 Swing 是 lightweight component, 而 AWT 是 heavyweight component。这是什么意思呢? 其实, 是指 Swing 是由纯 Java code 所写成的。因此 Swing 解决了 Java 因窗口类而无法跨平台的问题, 使窗口功能也具有跨平台与延展性的特性。而且 Swing 不需占有太多系统资源, 因此我们称 Swing 为 lightweight component, 表示我们可利用它轻易地做出各种变化。

相对于 Swing, 由于 AWT 具有 native code 的 C 语言成分, 所以若您想自行改动 AWT 的窗口变化时, 您必须编写自己的 C 语言 native code, 然后再搭配 AWT 的 native code 与 JDK 函数库。遇到不同的操作平台时, 又必须重新更改与编译自己所写的 native code。因此 AWT 不具有跨平台特性, 耗时且难以理解又耗费系统资源, 所以就称之为 heavyweight component, 表示 AWT 组件是不容易改动的。

2-1-2 您使用的 JDK 有包含 Swing 吗

Swing 在 Java1.1 的时期是属于外挂组件, 用户必须抓取 JFC 函数库 (Libraries), 设置 classpath 后, 才能使用 JFC 组件。但到了 JDK1.2 之后, JFC (with Swing) 已经包含在其中了, 用户可以直接在程序中声明和使用, 因此我们强烈建议读者使用 JDK1.2 以后的版本。在本书中的所有范例, 也是架构在 JDK1.2 以后的版本上。

Sun 将 JDK1.2 以后的版本称为 Java 2 系列。因此, JDK1.4 正确的说法应该是 Java 2 SDK, Standard Edition, v 1.4, 或是称为 Java 2 Standard Edition v1.4, 缩写为 J2SE1.4。笔者为方便说明, 将使用大家口头上常讲的 JDK1.4 或 J2SE1.4。



说明

JDK 是 Java Development Kit 的缩写, 而 J2SDK 是 Java 2 Software Development Kit 的缩写, 这两个意思是差不多的。另外还有一个缩写比较容易混淆, 但与前两者意思不同, 那就是 JSDK, 它是 Java Servlet Development Kit 的缩写, 读者应当在阅读时多注意这些缩写的意思, 以免搞混了!



注意

2-2 如何编译与运行包含 Swing 程序代码的 Java 程序

既然 Swing 是 Java 程序语言的一部分, 理所当然的, 我们必须先安装 JDK (Java Development Kit) 啰! 下面我们先来介绍到哪取得 JDK。

2-2-1 下载 JDK

欲下载 JDK, 请参考下列的步骤:

(1) 请由 “<http://java.sun.com/j2se/1.4/download.html>” 进到 JDK1.4 版下载网页。

在图 2-1 中请选择 Windows 版的 SDK。SDK 为 Standard Development Kits 的缩写, 即“标准开发工具”之义。

- (2) 接着看到的是一份如图 2-2 所示的软件授权声明，看完之后，请把滚动条移动到网页下方，点选“ACCEPT”按钮。

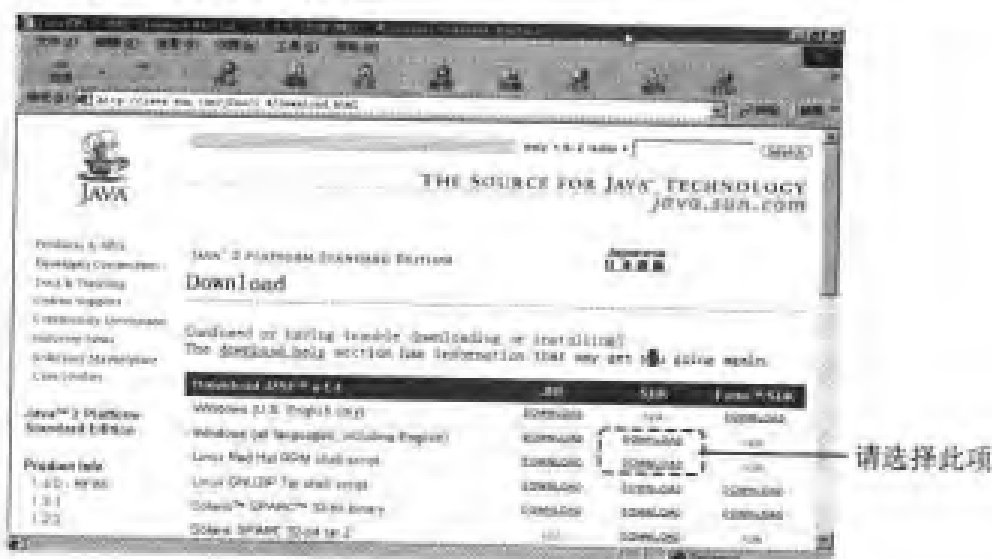


图 2-1

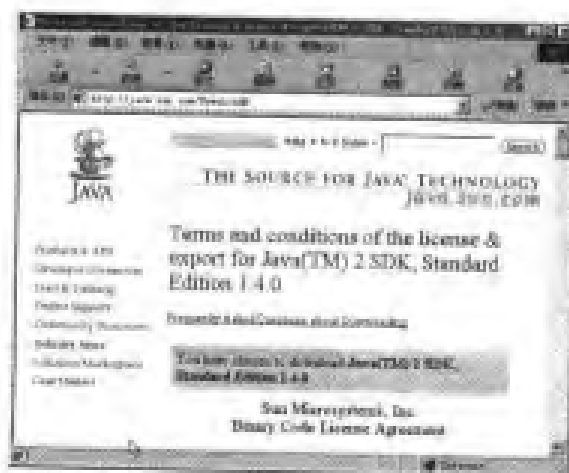


图 2-2

- (3) 接下来是下载文件的窗口，可选择离读者较近的站点下载文件，请点选按钮下载，如图 2-3 所示。

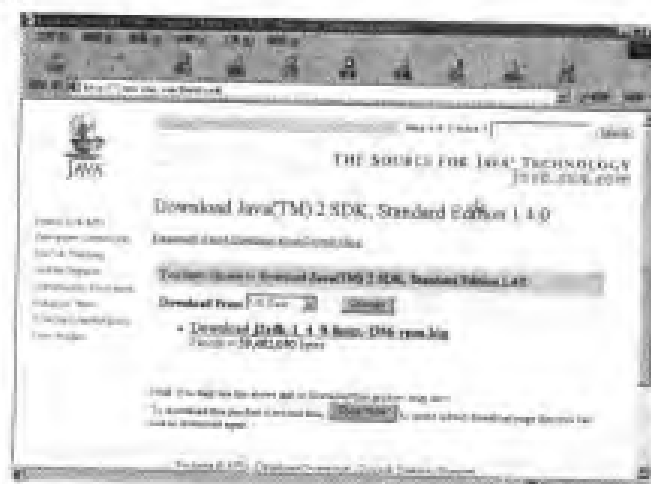


图 2-3

(4) 按下按钮下载之后, 会出现如图 2-4 所示的对话框, 按“确定”按钮进行文件下载。



图 2-4

此时会出现“另存为”窗口, 指定好欲存放的目录位置, 然后按下“保存”按钮后即可开始下载, 如图 2-5 所示。下载完成后, 就取得了 JDK1.4.0 版的安装文件。



图 2-5

2-2-2 安装 JDK

下载好 JDK 的文件之后, 即可着手进行安装, 请依下列的步骤来进行:

- (1) 在下载的文件图标上, 双击鼠标左键, 出现如图 2-6 所示的窗口后按“Next”按钮运行下一步。



图 2-6

- (2) 接下来会看到一份软件授权声明, 若同意这份授权内容, 可按“Yes”按钮继续, 如图 2-7 所示。

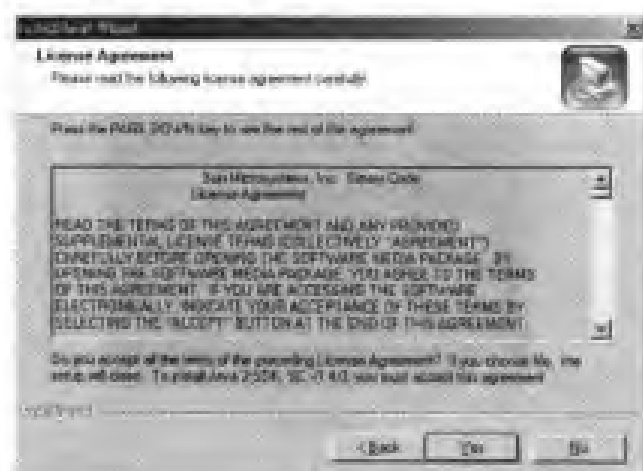
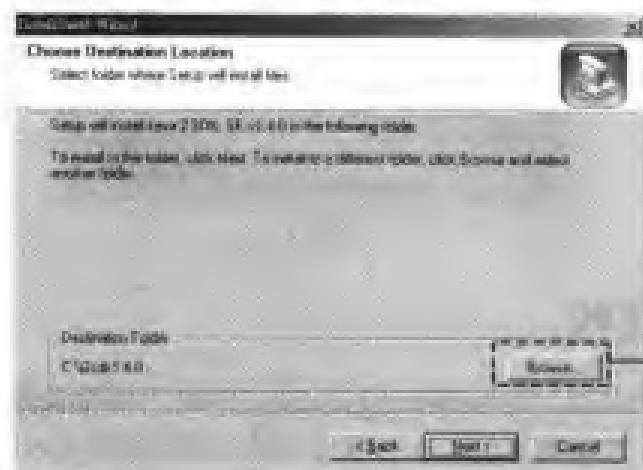


图 2-7

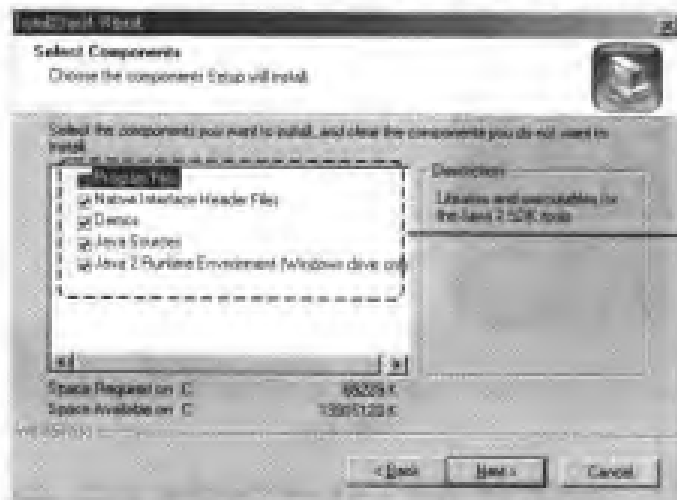
- (3) 接着要输入安装位置，请按“Browse”按钮选择欲安装的路径或使用默认值，建议使用直接使用默认的安装路径，也就是 C:\j2sdk1.4.0，如图 2-8 所示。



按下“Browse”按钮即可
选择其他安装路径

图 2-8

- (4) 选择好安装路径之后，按“Next”按钮出现以下画面，要求选择欲安装的项目。建议使用默认值，然后按“Next”按钮继续，如图 2-9 所示。



建议使用默认值

图 2-9

- (5) 接下来出现的画面询问要使用哪一种浏览器 (Browser) 做为 Java 的平台。可以选择“Microsoft Internet Explorer”或“Netscape 6”。笔者是用 Microsoft Internet Explorer 来上网, 因此选择它, 如图 2-10 所示。

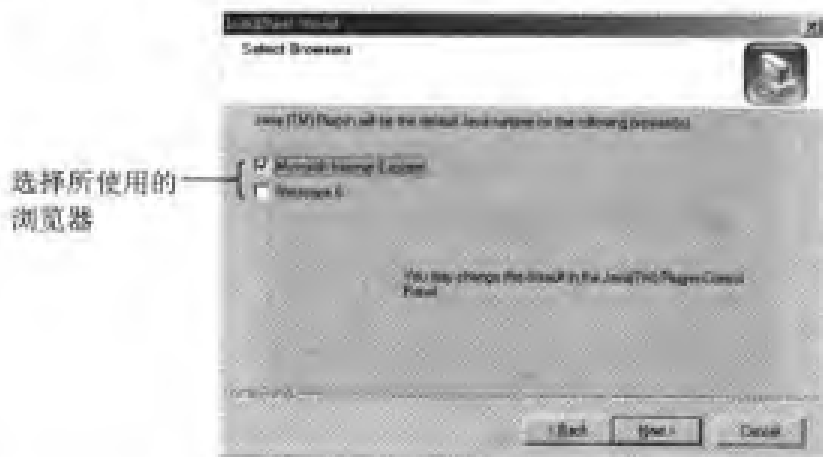


图 2-10

- (6) 接下来开始进行文件的复制与安装。待完成后, 会出现如图 2-11 所示的画面, 询问是否要读取 README 文件, 可以在方块中勾选, 以便阅读 README 的内容, 或直接按下“Finish”按钮结束安装程序。



图 2-11

2-2-3 设置 JDK 的操作环境

在使用 Java 来编译与运行程序之前, 必须先设置环境变量, 以方便系统知道 Java 的安装路径 (Path)。以下分两个小节来说明在不同的操作系统里如何设置环境变量:

◆ Windows XP/2000/NT 环境的设置

- (1) 打开“控制面板”, 找到“系统”图标, 双击鼠标左键即出现如图 2-12 所示的窗口, 请选择“高级”选项卡中的“环境变量”按钮。



图 2-12

- (2) 在“环境变量”窗口的“系统变量”字段里，先选择“Path”变量，再按下“编辑”按钮，此时出现“编辑系统变量”对话框。在“变量值”字段的最后面先输入一个分号“;”以区隔先前所设的路径，再输入“c:\j2sdk1.4.0\bin”，如图 2-13 所示。

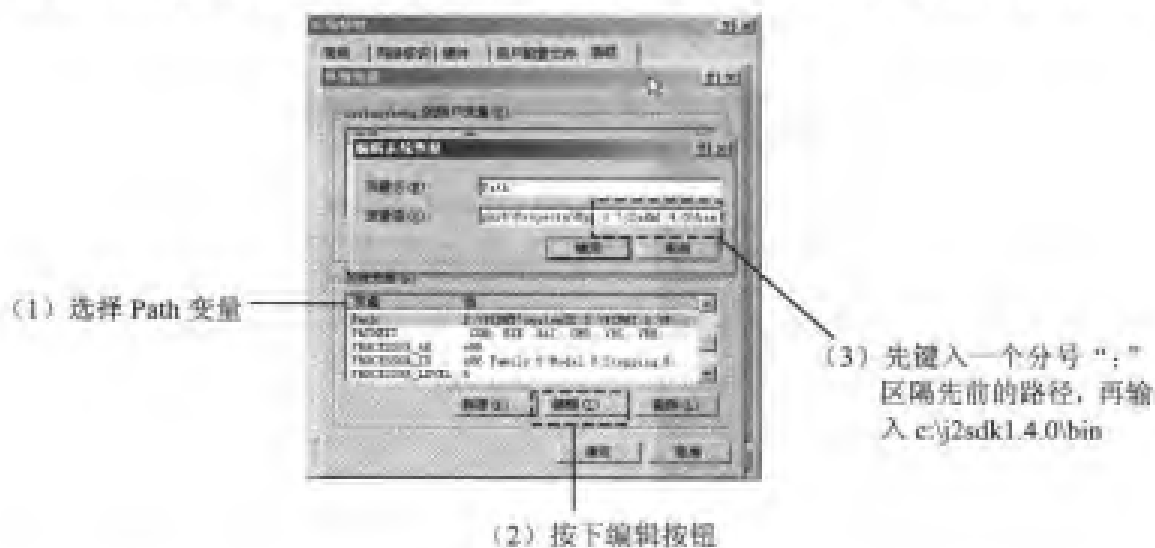


图 2-13

如果在“环境变量”窗口里找不到 Path 变量，则请按下“系统变量”字段里的“新建”按钮，在出现的“新建系统变量”对话框里填上如图 2-14 所示的内容。



图 2-14

设置完成后请按下“确定”按钮。

最后在“环境变量”窗口里按下“确定”按钮，再关闭“系统特性”窗口即完成路径的设置。

路径设置好之后，现在来测试一下路径是否已设置成功。请按一下窗口左下角的“开始”按钮，选择“程序”-“附件”-“命令提示符”打开 DOS 窗口，然后输入 java。



图 2-15

如果出现的画面与图 2-15 相同，恭喜用户已完成路径的设置。

❖ Windows 95/98/Me 环境的设置

- (1) 从“我的电脑”双击左键，點選 C 磁盘驱动器进入，接着选取窗口上方的“查看”菜单，再选择“文件夹选项”。
- (2) 在所出现的如图 2-16 所示的“文件夹选项”窗口里选择“查看”选项卡，找到“隐藏文件”里的“显示所有文件”，如果它没有被选取，请选取它，接着按“确定”按钮运行下一步骤。

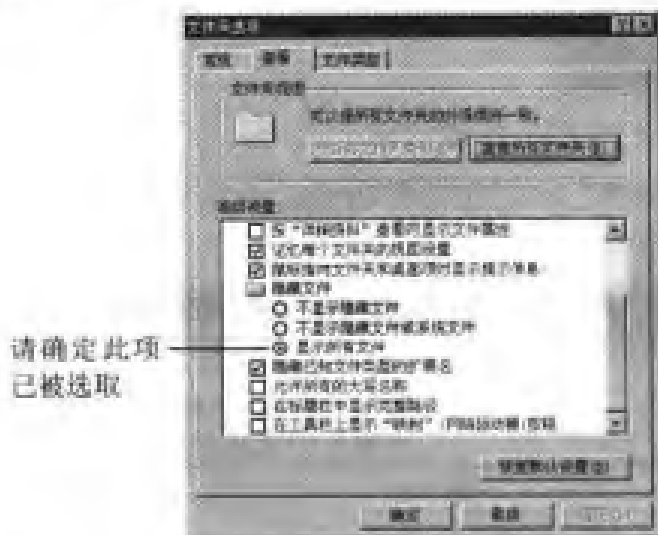


图 2-16

- (3) 在 C:\磁盘里找到“Autoexec.bat”这个文件，在图标上方按鼠标右键，选择“属性”，出现“Autoexec.bat 属性”窗口，将属性的“只读”勾选去除，再按下“确定”按钮，如图 2-17 所示。



图 2-17

- (4) 在“Autoexec.bat”图标上方按鼠标右键，在出现的菜单中选择“编辑”，此时 Windows 会用记事本打开它，如图 2-18 的窗口所示，当然 Autoexec.bat 内容会随着计算机操作环境的不同而会有些许的差异，但您应可找到“SET PATH”字符串。

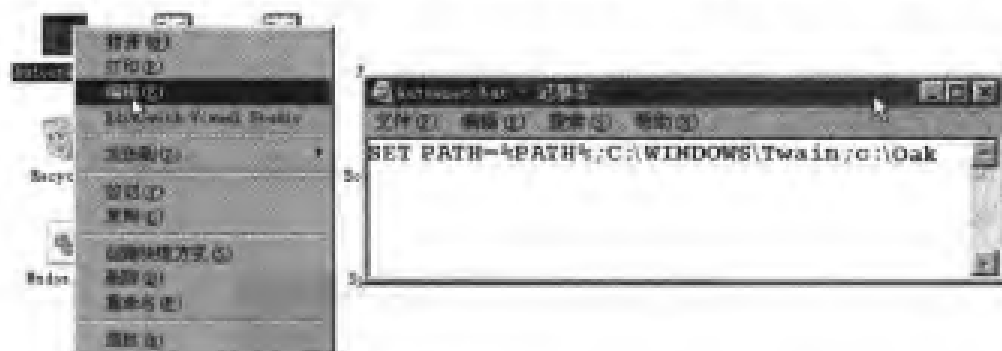


图 2-18

- (5) 找到“SET PATH”字符串之后，在它的最后面加上分号“:”，再输入“C:\jdk1.4.0\bin”字符串即可，如图 2-19 所示。

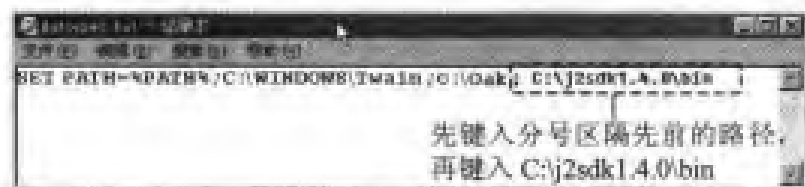


图 2-19

如果没有找到“SET PATH”字符串，请另起新的一行，再输入：

```
SET PATH= C:\jdk1.4.0\bin
```

- (6) 将 Autoexec.bat 存盘，重新开机之后，便可以使用 Java 了。

欲测试路径设置是否正确，请选择“程序”-“MS-DOS 模式”打开 DOS 窗口，然后输入 java，如图 2-20 所示。

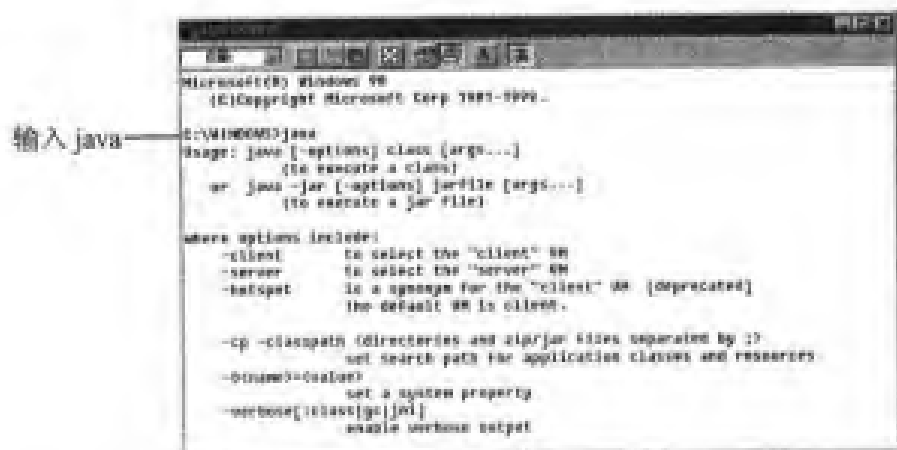


图 2-20

如果出现的画面与上图相同，代表路径已设置完成。

再次提醒读者，在本节路径的设置中，笔者是将 jdk1.4.0 安装在默认的 C:\jdk1.4.0 目录里，假使读者没将它安装在这个目录的话，请自行将“C:\jdk1.4.0\bin”修改成安装位置。

2-3 编写第一个 Java 程序

Java 程序可分为下列两种：（1）Java application；（2）应用在 WWW 上的 Java Applet。

Java Application 是指可以在 Java 平台上独立运行的一种程序，通常称之为 Java 应用程序，而 Java Applet 则是内嵌在 HTML 文件中，必须搭配浏览器来运行，因此有些人称 Applet 为网页向导，实不为过。Java Applet 的编写方式与 Java Application 类似，因此只要熟悉 Java Application 的编写方式，很快就能学会编写 Java Applet。本书除了第 19 章是介绍 Java Applet 的编写之外，其余章节所介绍的程序均是属于 Java Application。

在开始编写程序代码之前，请先在硬盘 C 中创建一个新的文件夹，并设文件夹名称为“Java”。本书所有的范例均假设存盘于 C:\Java 这文件夹下，如图 2-21 所示。



图 2-21 本书所有的范例均假设存盘于 C:\Java 文件夹里

2-3-1 编译与运行 Java Application

本节将开始练习编写 Java 程序了。首先介绍如何以最简单的方式来编写、编译与运行 Java Application。在编写程序之前，必须安装好 Java 开发工具 (Java Development Kit, JDK)，此时如果还没完成这个步骤，请参阅附录 A 先取得 JDK，再安装开发工具。

appl_1 是个相当简单的范例，经编译和执行后，它会在 DOS 窗口上显示 “Hello Java!!” 字符串。appl_1 的程序如下（注意每一行之前的行号是刻意加上去的，用以方便解说程序代码，它们并不是程序代码的一部分）：

```
01 // appl_1, 简单的 java application
02 public class appl_1
03 {
04     public static void main(String args[])
05     {
06         System.out.println("Hello Java !!");
07     }
08 }
```

现在把 appl_1 输入到记事本中，并把它存入到上一节所创建的 C:\Java 文件夹内，文件名设为 appl_1.java，如图 2-22 所示。

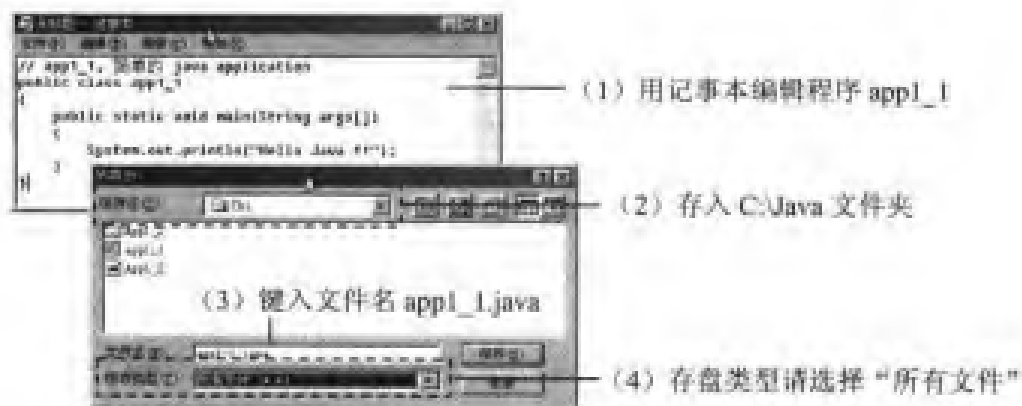


图 2-22 用记事本编写 Java 程序

记得在“另存为”对话框中，文件名应设为 appl_1.java，请勿用其他的名称，否则无法顺利编译。此外，记得在“保存类型”下拉列表内选择“所有文件”，如果此处选择“文本文件 (*.txt)”，将造成文件名称为 appl_1.java.txt，因而无法编译。

存好文件之后，接下来请依所使用的操作系统来打开 DOS 窗口：

❖ Windows XP/ 2000/NT:

选择“程序”-“附件”-“命令提示符”来打开 DOS 窗口。

❖ Windows Me/98/95:

选择“程序”-“MS-DOS 方式”来打开 DOS 窗口。

接下来请依下面的 3 个步骤来编译与运行 appl_1.java:

1. DOS 窗口打开后, 请先将文件夹切换到保存 appl_1.java 的 C:\Java 文件夹中, 即在 DOS 窗口内键入:

```
cd C:\Java
```

2. 切换到文件夹后, 请执行下面的命令来编译 appl_1.java:

```
javac appl_1.java
```

在上面的命令中, javac 是用来编译其后所接的 Java 程序, 它是由 java 与 c 合成的字, 而 c 正是 Compile (编译) 之意。

编译好了之后, 可以在 C:\Java 文件夹内发现一个文件名一样是 appl_1, 但扩展名为 “.class” 的文件。这个文件也就是前一节所说的 byte-codes, 如图 2-23 所示。



图 2-23 编译过后会产生文件名相同, 但扩展名为 “.class” 的文件

3. 编译好了之后, 请执行下面的命令来运行 byte-codes (即 appl_1.class):

```
java appl_1
```

注意在运行 byte-codes 时, 只需下达 “java <文件名>” 即可, 此处的文件名是指 byte-codes 的文件名, 但不能把 “.class” 也敲进去, 也就是说, 不能输入 “java appl_1.class” 来运行程序, 这样将会造成错误。

appl_1 的运行流程如图 2-24 所示:



图 2-24 appl_1 的运行流程, 是在 Windows 2000 的环境里运行的结果

这是最简单的 Java Application 程序, 本书所有的 Java Application 程序均可用上面的步骤进行编译与执行。

2-3-2 编译与运行 Java Applet

如前所述, Java Applet 内嵌于 HTML 文件里, 必须搭配浏览器来运行, 因此要运行 Java Applet, 必须要有 Applet 的 byte-codes 与支持 Java 的浏览器, 此时的浏览器即是扮演 Java 虚拟机 (JVM) 的角色, 用来解释 Java 的 byte-codes。

相同的, 还是介绍如何以最简单的方式来编写、编译与运行 Java Applet。提醒读者在编写程序之前, 必须安装好 Java 开发工具 (请参阅附录 A)。

App1_2.java 是个简单的 Applet, 它可在浏览器窗口上显示“Hello Java!!”字符串。App1_2 的程序如下:

```
01 // App1_2, Java Applet
02 import java.awt.*;
03 import java.Applet.*;
04 public class App1_2 extends Applet
05 {
06     public void paint(Graphics g)
07     {
08         g.drawString("Hello Java!!", 50, 50);
09     }
10 }
```

请仿照上一节的步骤, 用记事本编辑 App1_2.java, 并将它存盘在 C:\Java 文件夹里, 文件名为 App1_2.java。注意本书是以大写 A 开头的文件名代表 Applet 程序, 因此必须把文件名存成 App1_2.java, 而非 app1_2.java, 如图 2-25 所示。



图 2-25 App1_2.java 的内容

编译 Java Applet

编译 Java Applet 的步骤和编译 Java Application 的步骤完全相同, 但运行方式稍有不同, 先来编译 App1_2.java:

1. DOS 窗口打开后, 先将文件夹切换到 C:\Java, 即在 DOS 窗口内输入:

```
cd C:\Java
```

2. 切换好文件夹后, 执行下面的命令来编译 App1_2.java:

```
javac App1_2.java
```

编译好了之后, 可以在 C:\Java 文件夹内找到文件 App1_2.class。

◆ 准备好 HTML 文件

在前面曾说过,浏览器是扮演 Java 虚拟机 (JVM) 的角色,Applet 必须经过浏览器激活 Java 虚拟机才能执行程序,所以必须另外编辑一个 HTML 文件,在文件中指明 Applet 程序的文件名及路径,方便浏览器找到指定位置并下载 Applet 程序,其 HTML 文件内容如下所示:

```
01 <!-- Appl_2.htm -->
02 <HTML>
03 <APPLET CODE    = "Appl_2.class"
04             WIDTH = "200"
05             HEIGHT = "100" >
06             很抱歉,您的浏览器不支持 Java Applet!!
07 </APPLET>
08 </HTML>
```

其中<APPLET>...</APPLET>标签是用来赋值文件名称、路径和其他相关参数。CODE 参数用来指定 Applet 的路径和文件名,而 WIDTH 和 HEIGHT 用来指定 Applet 执行时显示在浏览器上的宽度及高度。如果浏览器不支持 Java,则<APPLET>与</APPLET>之间的文字会显示在浏览器上。

上面的 HTML 文件中,只赋值 CODE="Appl_2.class",并没有指明路径,因此必须把 HTML 文件和 Appl_2.class 存在同一个文件夹里。请先打开记事本,将 HTML 的语法编辑好,并将它保存在 C:\Java 文件夹下,文件名为 Appl_2.htm,如图 2-26 所示。



图 2-26 用记事本编写 HTML 文件,并存入 C:\Java 文件夹中

◆ 运行 Java Applet

编写好 Appl_2.htm 文件之后,要在浏览器里观看 Applet 就简单多了。首先打开 C:\Java 文件夹,找到 Appl_2.htm 文件之后,执行它即可看到 Applet 的运行结果,如图 2-27 所示。



图 2-27 运行 App1_2.htm 文件

如果使用 Windows XP 搭配 IE 6.0, 且从未下载过 JVM (Java 虚拟机), 则 XP 会跳出一窗口, 要求下载它, 如图 2-28 所示。

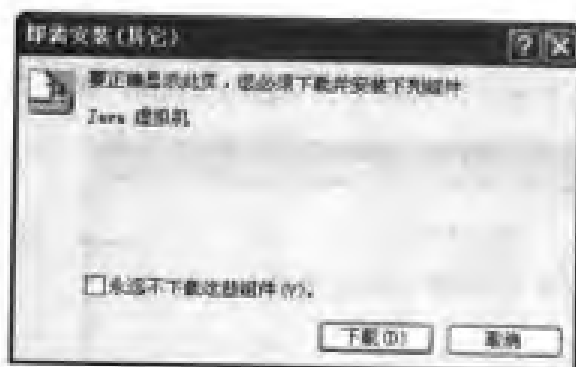


图 2-28 要求下载 JVM 的窗口。JVM 是被浏览器用来解释 Applet 的一种程序, 大小约 5MB

此时请按下“下载”按钮来下载, 下载完后便可以顺利地显示 Applet 了。当然, 如果这个下载 JVM 的窗口没有出现, 也可以到 Sun 的网站 <http://java.sun.com> 下载它。

❖ 利用 Appletviewer 程序运行 Java Applet

除了把浏览器当成 JVM 来运行 Applet 之外, JDK 也提供了 Appletviewer 程序, 可在不打开浏览器的情况下运行 Java Applet。

要以 Appletviewer 来观看 Applet 程序, 相同的, 必须先编译 Java 的原始文件成 byte-codes, 然后再准备好 HTML 文件, 这两个步骤与本节一开始所介绍的步骤均相同。

接下来确定路径已切换到存放.class 与.html 的文件夹下, 在本例中是“C:/Java”, 然后在 DOS 窗口内输入:

```
Appletviewer App1_2.html
```

按下【Enter】键就会看到“Applet 查看器”窗口与运行结果, 如图 2-29 所示。

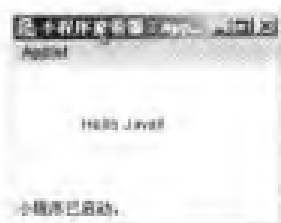


图 2-29 Appletviewer 运行的结果

本节已介绍了 Java 程序最基本的运行方式。本书所有的范例均可通过以上所介绍的步骤来运行。如果希望有个专属的集成环境（IDE）来编辑、编译与运行 Java 程序，可以参考附录 B——JCreator 的介绍。可以上网取得免费的 JCreator，它只有 2MB，但使用起来却相当的方便！

2-4 本章总结

在此章中，我们谈论了 Swing 的由来，JFC 的基本组件与功能，JRE 的概念与安装，运行 JAVA 程序可能产生的问题等等，让您不管在学习 Java Applet 或是 Java Application 时，都可以迎刃而解，而不会踏出第一步就灰心了。所谓好的开始是成功的一半，让我们一起来迈向成功的 Java Swing 之路吧！

2-5 本章习题

- (1) 简单说明 JFC 包含哪几种功能呢。
- (2) JDK 第几版以后将 Swing 纳入 JDK 的 Package 中？
- (3) 试述 Java Application 与 Java Applet 的不同。
- (4) 下面的程序在编译的过程中有什么问题？要如何修改呢？

```
01 import javax.swing.*;  
02 import java.awt.*;  
03 import java.awt.event.*;  
04  
05 public class SwingApplication {  
06  
07     public void aFrame() {  
08         JFrame F = new JFrame("Swing application");  
09         JLabel label = new JLabel("How are you??");  
10         F.getContentPane().add(label, BorderLayout.CENTER);  
11         F.pack();  
12         F.show();  
13         F.addWindowListener(new WindowAdapter() {  
14             public void windowClosing(WindowEvent e) {  
15                 System.exit(0);  
16             }  
17         });  
18     }  
19 }
```


3

使用版面管理器 (Layout Managers)

在此章中我们将介绍各种版面管理器的功能与运用。一个好的软件不仅在于它强大的功能、快速的效率等因素，设计一个良好的 (User Friendly) 用户界面，对软件本身而言也是一门学问，我们将带领读者了解 Java 所提供的各种版面管理器功能，并举出许多不同的程序范例以供读者参考。

陈入校

3-1 Swing 的版面结构

在我们谈论版面管理器之前，我们先来看看 Swing 的版面结构，这不仅可以让清楚地了解当我们放置组件至版面管理器时，应该注意什么事情，而且也能让我们充分运用版面管理器的功能。

Swing 中几乎所有组件都是从 JComponent 衍生（继承 JComponent）而来，也就是说这些组件都是 *lightweight Component*，均是由纯 Java Code 所编写而成，可对组件做最佳化操作来节省系统资源，并增加组件的使用弹性。至于 Swing 中哪些组件不是由 JComponent 衍生而来的呢？那就是 JFrame、JDialog、JWindow 与 JApplet，这四个组件是 *heavyweight Component*，必须使用到 *native code* 来画出这四个窗口组件，因为要在操作系统中显示窗口画面，必须使用操作系统的窗口资源，而以往的 AWT 组件大多使用 *native code* 所构造出来，因此 Swing 中的 JFrame 便继承原有 AWT 中的 Frame 类，而不是继承 JComponent 类。同样，JApplet 是继承原有 AWT 中的 Applet 类，也不是继承 JComponent 类。

JFrame、JDialog、JWindow 与 JApplet 这四个组件我们统称为最上层（Top-Level）组件，因为其余的 Swing 组件都必须依附在此四组件之一上才能显示出来。此四组件均能实现（Implement）RootPaneContainer 这个界面（Interface），此界面定义了各种容器取得与设置的方法，这里的容器包括 JRootPane、Glass Pane、Layered Pane 和 Content Pane。其中 JRootPane 并不是真实的容器，它是由 Glass Pane 与 Layered Pane 所组成（Layered Pane 里拥有 Content Pane 与 Menu Bar，而 Menu Bar 可选择使用或不使用），我们不能在 JRootPane 上加入任何的组件，因为它只是一个虚拟的容器，若要在最上层组件上加入组件，就必须加在 Layered Pane 或是 Layered Pane 里的 Content Pane 上。以 JFrame 为例，一般我们要在 JFrame 上加入其他组件（如 JButton 或 JLabel 等）必须先取得 JFrame 的 Content Pane，然后将要加入的组件放在 Content Pane 中，而不是直接就加到 JFrame 上。因此若要在 JFrame 中加入一个按钮，不能像以前 AWT 时一样写成 `frame.add(button)` 的形式，而必须先取得 JFrame 的 Content Pane，然后将按钮加入 Content Pane 中，如：

```
frame.getContentPane().add(button)
```

否则在编译的时候将有错误信息产生。

1. 组件必须加在容器中，而容器本身具有层次性的关系，就如同珠宝盒一般，大盒子里面可以放小盒子，小盒子里面还可以放更小的盒子，而珠宝就可以放在某一个盒子中，这里的珠宝就代表组件，盒子就代表容器。因此若您想在 JFrame 中加入任何组件时，必须先取得 JFrame 的容器来放置这些组件，而由于 JFrame、JDialog、JWindow 与 JApplet 是显示 Swing 组件的源头，我们可以称它们为根组件，也就是所谓的最上层（Top-Level）组件。
2. RootPaneContainer 它是一个 Interface，共有 5 个类实现（Implement）它，分别是 JFrame、JApplet、JWindow、JDialog、JInternalFrame，其中 JInternalFrame 是一个 *lightweight Component*，它不是一个最上层（Top-Level）组件，也就是说 JInternalFrame 不能单独显示出来，必须依附在最上层组件中，我们将


注意

在其他章节讨论此组件。而 JFrame、JApplet、JWindow、JDialog 均为最上层 (Top-Level) 组件。

RootPaneContainer 定义了下面几种方法，如表 3-1 所示。

表 3-1

返回类型	方 法 名
Container	getContentPane() 返回 ContentPane
Component	getGlassPane() 返回 GlassPane
JLayeredPane	getLayeredPane() 返回 LayeredPane
JRootPane	getRootPane() 返回属于这个组件的 JRootPane
Void	setContentPane(Container contentPane) 设置 ContentPane
Void	setGlassPane(Component glassPane) 设置 GlassPane
Void	setLayeredPane(JLayeredPane layeredPane) 设置 LayeredPane

我们来看 JFrame 取得 Content Pane 的实际流程，下面是一段很简单的程序代码：

```
public class Simple{
    Simple(){
        JFrame frame = new JFrame();
        Container contentPane = frame.getContentPane();
        JButton button = new JButton();
        contentPane.add(button);
    }
}
```

当我们写 `frame.getContentPane()` 时，会返回此 `frame` 的 Content Pane，也就是一个容器 (Container) 组件，有了容器之后我们才
能将 `button` 组件摆进去，此时 `JFrame` 才算拥有 `button` 组件。所以 `JFrame` 就好像是一块空地，要在这块空地上住人应该先盖一栋房子（容器），然后人、家具、设备等（组件）就能搬进此房子中了。图 3-1 说明了最上层组件都含有 `JRootPane` 组件，`JRootPane` 本身就含有容器组件，可

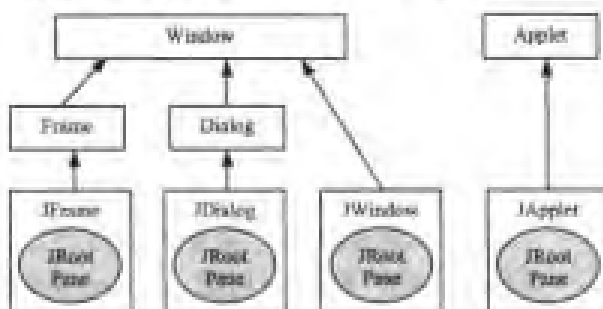


图 3-1

让最上层组件装入其他的组件。

由此图可以很清楚地看出, `JFrame`、`JDialog`、`JWindow` 是继承 `Window` 类而来, 而 `JApplet` 是继承 `Applet` 这个类, 他们均含有一个相同的子组件, 那就是 `JRootPane`。接下来, 我们以图 3-2 来表示 `JRootPane` 的组成结构, 让读者印象更为深刻也更容易了解。

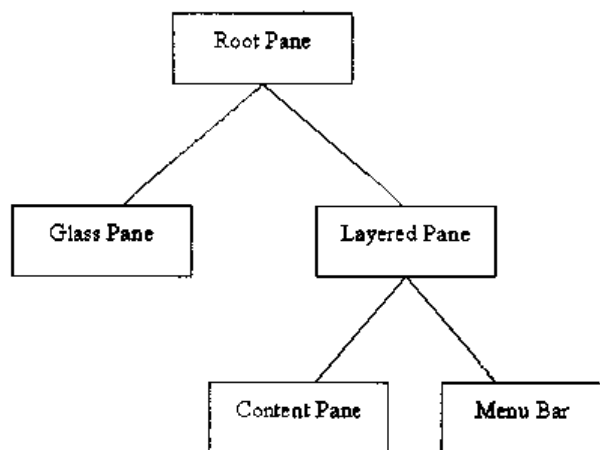


图 3-2

由上图中我们可看到 `JRootPane` 包含有 `Glass Pane` 与 `Layered Pane`, 而 `Layered Pane` 又包含 `Content Pane` 与 `Menu Bar`, 其中程序设计者可选择是否使用 `Menu Bar`, 若没有使用 `Menu Bar`, 则 `Content Pane` 就会占据整个版面。因此当您建立一个 `JFrame` 组件时, 系统会为此 `JFrame` 建立 `Root Pane` (称为 `JRootPane`) 组件, 您就能任意地取得 `Root Pane` 上的 `Glass Pane`、`Layered Pane` 或 `Content Pane`, 然后进行其他的操作。至于这 3 个组件有什么作用, 我们来看下面的说明:

- ✿ **Glass Pane:** 顾名思义它是一个透明的面版, 主要功能是它可以捕获 `JFrame` 上的任何事件 (Event), 例如鼠标在 `JFrame` 上的任何操作。 `Glass Pane` 的默认值是不可看见的, 不过您也可以将它设为可看见的 (Visible), 例如若您要在 `Glass Pane` 上做绘图的工作, 就可以在 `Glass Pane` 上显示出来。
- ✿ **Layered Pane:** 它是一个可让您重叠组件的面版, 本身也是一个容器, 可直接加入组件在此容器中。例如当您在 `JFrame` 上看到 `Menu Bar` 或 `Popup Menu`, 或者其他相互重叠的组件, 就是使用 `Layered Pane` 的效果。您可以将 `Layered Pane` 看成是相当多层的置物架, 每一层都可以放置物品, 而且上面的物品会遮住下面的物品。
- ✿ **Content Pane:** 是最重要, 且是最常用到的容器。它是 `Layered Pane` 中的一层, 一般均视为最底层。通常我们会将组件置于最上层组件的 `Content Pane` 上, 而不会加在 `Layered Pane` 中, 因为若将组件加入 `Layered Pane` 中必须自行管理组件间层次的关系, 这无疑会加重程序维护的困难度 (此部分将在第 5 章中详加说明)。若将组件加入 `Content Pane` 中, 对于像工具栏 (Tool Bar)、`Popup Menu` 等组件, Java 会自动调整其层次关系, 完全无需程序设计人员来伤脑筋。因此要将组件加入 `JFrame` 或 `JApplet` 中, 我们会利用 `getContentPane()` 方法取得 `Content Pane` 容器, 再利用 `add()` 方法将组件加到 `Content Pane` 中。现在, 您对下面这行程序应该不陌生了吧:

```
frame.getContentPane().add(button)
```

3-2 版面管理器 (Layout Manager)

一个软件的成功与否，不仅在于软件本身的效率、稳定性，用户界面 (UI) 也是一项非常重要的因素。因此，如何设计一个良好的 (User Friendly) 用户界面，正是本章的重点。所谓版面管理器正是管理那些放进版面的组件的，例如每年过年我们都会有大清扫的工作，在大清扫时都会将客厅的东西先搬出去，等清扫完毕之后再将这些东西重新摆设回去。如果我们有一个机器人可以为我们搬移这些东西，我们应只需要跟它说电视该放哪、沙发该放哪、摆饰柜该放哪就行了，剩下搬动的工作机器人就会帮我们完成。因此版面管理器就如同上例的机器人一般，你只要告诉他东西该怎么摆，他就会把它记起来，并且把东西摆在你要摆的位子上。

使用版面管理器 (Layout Manager) 除了让您的组件有秩序地摆放在适当的位置上外，另一个好处就是当您改动窗口大小时，版面管理器会自动更新您的版面来配合窗口大小，不需要担心版面会因此乱掉。

3-2-1 Layout Manager 的种类与介绍

在 Java 中共定义了这几个 Layout Manager 可供您使用，如下所示：

1. BorderLayout
2. FlowLayout
3. GridLayout
4. CardLayout
5. GridBagLayout
6. BoxLayout

上面的 1~5 项 Layout 是在 AWT 时已有的 Layout Manager，而第 6 项是 Swing 所提供新的 Layout Manager。

其实 Swing 中还有三个 Layout，那就是 ScrollPaneLayout、ViewportLayout 与 OverlayLayout，但由于这三个 Layout 系统会自行管理，在大部分的情况下用户并不需要知道如何使用，因此我们不对这三种 Layout 多作介绍。



说明

3-2-2 BorderLayout 的使用

BorderLayout 的类层次结构图：

```
java.lang.Object
    --java.awt.BorderLayout
```

BorderLayout 将版面划分成东、西、南、北、中五个区域，您可以设置要将组件放到这五个区域中的哪个区域中。要使用 BorderLayout 相当简单，只要在容器中设置使用 BorderLayout 这种版面管理即可，表 3-2 为 BorderLayout 的构造函数。

表 3-2

构造函数

BorderLayout()

建立一个组件间没有间距的 Border Layout

BorderLayout(int hgap, int vgap)

建立一个组件间有间距的 Border Layout

我们来看一个例子，您就很容易了解了：

范例 BorderLayoutDemo.java (文件位于随书光盘目录
exam\ch3\BorderLayoutDemo.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class BorderLayoutDemo {
6
7      public BorderLayoutDemo() {
8
9          JFrame f = new JFrame();
10
11          Container contentPane = f.getContentPane();
12          contentPane.setLayout(new BorderLayout());
13          contentPane.add(new JButton("EAST"),BorderLayout.EAST);
14          contentPane.add(new JButton("WEST"),BorderLayout.WEST);
15          contentPane.add(new JButton("SOUTH"),BorderLayout.SOUTH);
16          contentPane.add(new JButton("NORTH"),BorderLayout.NORTH);
17          contentPane.add(new
18              JLabel("CENTER",JLabel.CENTER),BorderLayout.CENTER);
19
20          f.setTitle("BorderLayout");
21          f.pack();
22          f.setVisible(true);
23
24          f.addWindowListener(new WindowAdapter() {
25              public void windowClosing(WindowEvent e) {
26                  System.exit(0);
27              }
28          });
29      }
30
31      public static void main(String args[]) {
32
33          BorderLayoutDemo b = new BorderLayoutDemo();
34
35      }
36  }

```

⊕ 说明：

- (1) 由于这是一个 Application 程序，因此我们在第 9 行产生一个 Frame 当作 Top-Level 组件。

- (2) 11~12 行在取得此 Frame 的 ContentPane, 并设置它为 BorderLayout。
- (3) 13~18 行是将各组件加入 ContentPane 中, 并指定其相关位置。读者要注意, 当您没有设置组件相关位置时, BorderLayout 将以 CENTER 当作默认值, 若有两个组件安置在相同位置, 将可能产生位置错乱的情况, 因此在写程序时, 应当养成习惯写清楚组件摆设位置, 不仅可避免位置错乱的情况产生, 也使程序容易理解并容易检测。
- (3) 24~28 行是在处理关闭窗口的操作, 若您没写这一段, 就算您已经关闭窗口了, 但程序并不会终止。

编写好之后, 我们运行编译与运行的操作:

```
javac BorderLayoutDemo.java  
java BorderLayoutDemo
```

我们可以看到如图 3-3 所示的运行结果:



图 3-3

设置组件的间距, 您可以使用有间距参数的 BorderLayout 构造函数, 也可以利用 BorderLayout 的 setHgap (int hgap) 与 setVgap (int vgap) 两个方法来达成。

所以, 若我们将上一个程序的第 12 行改成 contentPane.setLayout(new BorderLayout(10,10)), 则运行后的结果将如图 3-4 所示, 各组件间将有间距存在:



图 3-4

3-2-3 FlowLayout 的使用

FlowLayout 的类层次结构图:

```
java.lang.Object  
--java.awt.FlowLayout
```

FlowLayout 是一个相当简单的排列方法, 就如同流水一般, FlowLayout 将组件一个接着一个往右一直排列下去, 若组件个数太多, 无法在同一行显示时, FlowLayout 会自动将组件

往下一行排列。要使用 `FlowLayout` 只要在容器中设置使用 `FlowLayout` 这种版面管理即可，下面为 `FlowLayout` 的构造函数，如表 3-3 所示。

表 3-3

构造函数	
<code>FlowLayout()</code>	建立一个新的 <code>Flow Layout</code> ，此 <code>FlowLayout</code> 默认值是居中对齐，组件彼此有 5 单位的水平与垂直间距
<code>FlowLayout(int align)</code>	建立一个新的 <code>Flow Layout</code> ，此 <code>FlowLayout</code> 可设置排列方式，组件彼此有 5 单位的水平与垂直间距
<code>FlowLayout(int align, int hgap, int vgap)</code>	建立一个新的 <code>Flow Layout</code> ，此 <code>FlowLayout</code> 可设置排列方式与组件间距

我们来看一个实际的例子：

范例 `FlowLayoutLayoutDemo` (文件位于随书光盘目录
examich3\FlowLayoutLayoutDemo.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class FlowLayoutDemo {
6
7      public FlowLayoutDemo() {
8
9          JFrame f = new JFrame();
10
11          Container contentPane = f.getContentPane();
12          contentPane.setLayout(new FlowLayout());
13          contentPane.add(new JButton("first"));
14          contentPane.add(new JButton("second"));
15          contentPane.add(new JButton("third"));
16          contentPane.add(new JButton("fourth"));
17          contentPane.add(new JButton("fifth"));
18          contentPane.add(new JButton("This is the last button"));
19
20          f.setTitle("FlowLayout");
21          f.setSize(getPreferredSize());
22          f.setVisible(true);
23
24          f.addWindowListener(new WindowAdapter() {
25              public void windowClosing(WindowEvent e) {
26                  System.exit(0);
27              }
28          });
29      }
30
31      public Dimension getPreferredSize()
32      {
33          return new Dimension(200, 200);
34      }
35

```

```
36     public static void main(String args[]) {  
37  
38         FlowLayoutDemo b = new FlowLayoutDemo();  
39  
40     }  
41 }
```

说明：

- (1) 由于这是一个 Application 程序，因此我们在第 9 行产生一个 Frame 当作 Top-Level 组件。
- (2) 11~12 行在取得此 Frame 的 ContentPane，并设置它为 FlowLayout。
- (3) 13~18 行是将各组件依次加入 ContentPane 中。
- (4) 第 21 行是设置窗口出现的大小，注意，这里必须将 f.pack() 这行拿掉，否则 setSize 功能将没有作用。
- (5) 24~29 行是在处理关闭窗口的操作。

编写好之后，我们运行编译与运行的操作：

```
javac FlowLayoutDemo.java  
java FlowLayoutDemo
```

我们可以看到如图 3-5 所示的运行结果：



图 3-5

您可以使用有间距参数的 FlowLayout 构造函数，使 FlowLayout 的排列具有间距，并可利用排列方向参数来指定靠什么方向排列。FlowLayout 共有五种排列方式，依次是 CENTER（默认值）、LEFT、RIGHT、LEADING、TRAILING，若我们将上一个程序的第 12 行改成 `contentPane.setLayout(new FlowLayout(FlowLayout.LEFT))`，则结果将如图 3-6 所示。



图 3-6

3-2-4 GridLayout 的使用

GridLayout 的类层次结构图:

```
java.lang.Object
    --java.awt.GridLayout
```

GridLayout 比 FlowLayout 多了行和列的设置,也就是说你要先设置 GridLayout 共有几行几列,就如同二维平面一般,然后您加进去的组件会先填完第一行的格子,然后再从第二行开始填,依此类推,就像是一个个的格子一般。而且 GridLayout 会将所填进去组件的大小设为一样,表 3-4 是 GridLayout 的构造函数。

表 3-4

构造函数

GridLayout()	建立一个新的 GridLayout, 默认值是 1 行 1 列
GridLayout(int rows, int cols)	建立一个几行几列的 GridLayout
GridLayout(int rows, int cols, int hgap, int vgap)	建立一个几行几列的 GridLayout, 并设置组件的间距

我们实际来看一个例子:

范例 GridLayoutDemo.java (文件位于随书光盘目录
exam\ch3\GridLayoutDemo.java)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class GridLayoutDemo {
6
7      public GridLayoutDemo() {
8
9          JFrame f = new JFrame();
10
11          Container contentPane = f.getContentPane();
12          contentPane.setLayout(new GridLayout(6,1));
13          contentPane.add(new JButton("first"));
14          contentPane.add(new JButton("second"));
15          contentPane.add(new JButton("third"));
16          contentPane.add(new JButton("fourth"));
17          contentPane.add(new JButton("fifth"));
18          contentPane.add(new JButton("This is the last button"));
19
20          f.setTitle("GridLayout");
21          f.pack();
22          f.setVisible(true);
23
24          f.addWindowListener(new WindowAdapter() {
25              public void windowClosing(WindowEvent e) {
26                  System.exit(0);
27              }
28          });
29      }
30  }
```

```

28         });
29     }
30
31     public static void main(String args[]) {
32
33         GridLayoutDemo b = new GridLayoutDemo();
34
35     }
36 }

```

⊕ 说明:

- (1) 由于这是一个 Application 程序, 因此我们在第 9 行产生一个 Frame 当作 Top-Level 组件。
- (2) 11~12 行在取得此 Frame 的 ContentPane, 而且设置它为 6 行 1 列的 GridLayout。
- (3) 13~18 行是将各组件加入 ContentPane 中, GridLayout 会依次将组件摆入其中。
- (4) 24~28 行是在处理关闭窗口的操作, 若您没写这一段, 就算您已经关闭窗口了, 但程序并不会终止。

编写好了以后, 我们运行编译与运行的操作:

```
javac GridLayoutDemo.java
```

```
java GridLayoutDemo
```

我们可以看到如图 3-7 所示的运行结果:



图 3-7

上图中, GridLayout 将组件的大小化整为一, 并依我们的指示将组件排为 6 行 1 列。若我们在程序的第 12 行改成 `contentPane.setLayout(new GridLayout(5,1))`, 会是什么结果呢? 由于我们共有 6 个组件, 但我们却是设置为 5 行 1 列, 因此最后一个 button 没有空间给他摆设, 在这种情况下, 系统在显示 JFrame 时会自动将版面变为 5 行 2 列, 以便容纳多余的组件, 如图 3-8 所示。

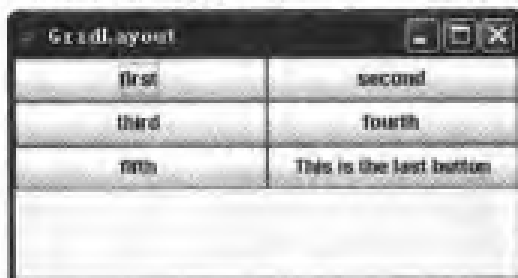


图 3-8

您可以使用有间距参数的 GridLayout 构造函数, 建立出具有间距的 GridLayout。

3-2-5 CardLayout 的使用

CardLayout 的类层次结构图:

```
java.lang.Object
    --java.awt.CardLayout
```

CardLayout 的功能就如同你有很多张卡片叠在一起,你一次只能看到其中一张卡片,但你可以任意地抽出其中一张卡片来看。表 3-5 是 CardLayout 的构造函数:

表 3-5

构造函数
CardLayout() 建立一个新的 Card Layout, 组件间没有水平与垂直间距
CardLayout(int hgap, int vgap) 建立一个新的 Card Layout, 可指定组件间的水平与垂直间距

我们直接来看个例子您就很容易了解了。

范例 CardLayoutDemo.java (文件位于随书光盘目录
exam\ch3\CardLayoutDemo.java)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class CardLayoutDemo implements ActionListener{
6
7      JPanel p1,p2,p3,p4;
8      int i = 1;
9      JFrame f;
10
11     public CardLayoutDemo() {
12
13         f = new JFrame();
14         Container contentPane = f.getContentPane();
15         contentPane.setLayout(new GridLayout(2,1));
16
17         p1 = new JPanel();
18         Button b = new Button("Change Card");
19         b.addActionListener(this);
20         p1.add(b);
21         contentPane.add(p1);
22
23         p2 = new JPanel();
24         p2.setLayout(new FlowLayout());
25         p2.add(new JButton("first"));
26         p2.add(new JButton("second"));
27         p2.add(new JButton("third"));
28         p3 = new JPanel();
29         p3.setLayout(new GridLayout(3,1));
30         p3.add(new JButton("fourth"));
31         p3.add(new JButton("fifth"));
```

```

32         p3.add(new JButton("This is the last button"));
33         p4 = new JPanel();
34         p4.setLayout(new CardLayout());
35         p4.add("one",p2);
36         p4.add("two",p3);
37         ((CardLayout)p4.getLayout()).show(p4,"one");
38
39         contentPane.add(p4);
40
41         f.setTitle("CardLayout");
42         f.pack();
43         f.setVisible(true);
44
45         f.addWindowListener(new WindowAdapter() {
46             public void windowClosing(WindowEvent e) {
47                 System.exit(0);
48             }
49         });
50     }
51
52     public void actionPerformed(ActionEvent event)
53     {
54         switch(i)
55         {
56             case 1:
57                 ((CardLayout)p4.getLayout()).show(p4,"two");
58                 break;
59             case 2:
60                 ((CardLayout)p4.getLayout()).show(p4,"one");
61                 break;
62         }
63         i++;
64         if (i==3)
65             i=1;
66
67         f.validate();
68     }
69
70     public static void main(String args[]) {
71
72         new CardLayoutDemo();
73
74     }
75 }

```

⊕ 说明:

- (1) 由于这是一个 Application 程序,因此我们在第 13 行产生一个 Frame 当作 Top-Level 组件。
- (2) 在 14~15 行取得此 Frame 的 ContentPane, 并设置它为 2 行 1 列的 GridLayout, 第一行是放含有“Change Card”按钮的 JPanel, 第二行则是要放 CardLayout 的 JPanel。

- (3) 第 19 行是指说当按下“Change Card”按钮时，将会有系统操作产生。而系统将做什么操作，我们写在 52~68 行。
- (4) 23~27 行我们产生一个 JPanel，设置成 FlowLayout，并加入 3 个按钮。
- (5) 28~32 我们产生另一个 JPanel，设置成 GridLayout，并加入 3 个按钮。
- (6) 33~36 我们产生最后一个 JPanel，将它设置成 CardLayout，目的是将上面两个 JPanel 加入此 JPanel 中。
- (7) 因为 CardLayout 是管理一堆卡片，既然如此，要 show 出哪一张卡片，就必须使用 show(Container parent, String name) 这个方法，第一个参数是指管理卡片的 Container，在此程序中就是指 p4，第二个参数是指要显示出的卡片名称，如第 37 行所示。
- (8) 第 39 行是将 CardLayout 的 JPanel 加入 contentPane 中。
- (9) 45~49 行是在处理关闭窗口的操作。

编写好之后，我们先来运行编译与运行的操作：

```
javac CardLayoutDemo.java  
java CardLayoutDemo
```

我们可以看到如图 3-9 所示的运行结果：



图 3-9

当我们按下“Change Card”按钮时，下面的组件会被置换成如图 3-10 所示。



图 3-10

再按一次“Change Card”按钮时，下面的组件又会回到原来的样子，因为我们现在只有两张卡片（Card）。

要显示 CardLayout 的卡片，除了用 show(Container parent, String name) 这个方法以外，读者还可以试试 first(Container)、next(Container)、previous(Container)、last(Container) 这四个方法，一样可以达到显示的效果。

3-2-6 GridBagLayout 的使用

GridBagLayout 的类层次结构图:

```
java.lang.Object
    --java.awt.GridBagLayout
```

GridBagLayout 是 Java 中最有弹性但也是最复杂的一种版面管理器, 在之前我们所介绍的版面管理器中, 我们不能设置每个组件该占多少版面空间, 只能设置组件该放那个位置, 而在 GridBagLayout 里, 不仅能设置组件摆设的位置, 也能设置组件的大小。GridBagLayout 只有一种构造函数, 但 GridBagLayout 必须配合 GridBagConstraints 才能达到设置的效果, 表 3-6 是这两个类的构造函数:

表 3-6

构造函数
GridBagLayout() 建立一个新的 GridBagLayout 管理器
GridBagConstraints() 建立一个新的 GridBagConstraints 对象
GridBagConstraints(int gridx, int gridy, int gridwidth, int gridheight, double weightx, double weighty, int anchor, int fill, Insets insets, int ipadx, int ipady) 建立一个新的 GridBagConstraints 对象, 并指定其参数的值

我们现在来看一个实际的例子吧!

范例 GridBagLayoutDemo.java (文件位于随书光盘目录 exam1ch3\GridBagLayoutDemo.java)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class GridBagLayoutDemo {
6
7      public GridBagLayoutDemo() {
8
9          JButton b;
10         GridBagConstraints c;
11         int gridx, gridy, gridwidth, gridheight, anchor, fill, ipadx, ipady;
12         double weightx, weighty;
13         Insets inset;
14
15         JFrame f = new JFrame();
16
17         GridBagLayout gridbag = new GridBagLayout();
18         Container contentPane = f.getContentPane();
19         contentPane.setLayout(gridbag);
20
21         b= new JButton("first");
22         gridx=0;
23         gridy=0;
24         gridwidth = 1;
```

```
25         gridheight = 1;
26         weightx = 10;
27         weighty = 1;
28         anchor = GridBagConstraints.CENTER;
29         fill = GridBagConstraints.HORIZONTAL;
30         inset = new Insets(0,0,0,0);
31         ipadx = 0;
32         ipady = 0;
33         c = new GridBagConstraints(gridx,gridy,gridwidth,gridheight,
34 weightx,weighty,anchor,fill,inset,ipadx,ipady);
35         gridbag.setConstraints(b,c);
36         contentPane.add(b);
37
38         b= new JButton("second");
39         gridx=1;
40         gridy=0;
41         gridwidth = 2;
42         gridheight = 1;
43         weightx = 1;
44         weighty = 1;
45         anchor = GridBagConstraints.CENTER;
46         fill = GridBagConstraints.HORIZONTAL;
47         inset = new Insets(0,0,0,0);
48         ipadx = 50;
49         ipady = 0;
50         c = new GridBagConstraints(gridx,gridy,gridwidth,gridheight,
51 weightx,weighty,anchor,fill,inset,ipadx,ipady);
52         gridbag.setConstraints(b,c);
53         contentPane.add(b);
54
55         b= new JButton("third");
56         gridx=0;
57         gridy=1;
58         gridwidth = 1;
59         gridheight = 1;
60         weightx = 1;
61         weighty = 1;
62         anchor = GridBagConstraints.CENTER;
63         fill = GridBagConstraints.HORIZONTAL;
64         inset = new Insets(10,0,0,0);
65         ipadx = 0;
66         ipady = 50;
67         c = new GridBagConstraints(gridx,gridy,gridwidth,gridheight,
68 weightx,weighty,anchor,fill,inset,ipadx,ipady);
69         gridbag.setConstraints(b,c);
70         contentPane.add(b);
71
72         b= new JButton("fourth");
73         gridx=1;
74         gridy=1;
75         gridwidth = 1;
76         gridheight = 1;
77         weightx = 1;
78         weighty = 1;
79         anchor = GridBagConstraints.CENTER;
```

```

80         fill = GridBagConstraints.HORIZONTAL;
81         inset = new Insets(0,0,0,0);
82         ipadx = 0;
83         ipady = 0;
84         c = new GridBagConstraints(gridx,gridy,gridwidth,gridheight,
85 weightx,weighty,anchor,fill,inset,ipadx,ipady);
86         gridbag.setConstraints(b,c);
87         contentPane.add(b);
88
89         b= new JButton("This is the last Button");
90         gridx=2;
91         gridy=1;
92         gridwidth = 1;
93         gridheight = 2;
94         weightx = 1;
95         weighty = 1;
96         anchor = GridBagConstraints.SOUTH;
97         fill = GridBagConstraints.HORIZONTAL;
98         inset = new Insets(0,0,0,0);
99         ipadx = 0;
100        ipady = 0;
101        c = new GridBagConstraints(gridx,gridy,gridwidth,gridheight,
102 weightx,weighty,anchor,fill,inset,ipadx,ipady);
103        gridbag.setConstraints(b,c);
104        contentPane.add(b);
105
106        f.setTitle("GridBagLayout");
107        f.pack();
108        f.setVisible(true);
109
110        f.addWindowListener(new WindowAdapter() {
111            public void windowClosing(WindowEvent e) {
112                System.exit(0);
113            }
114        });
115    }
116
117    public static void main(String args[]) {
118
119        new GridBagLayoutDemo();
120    }
121 }

```

由于 GridBagLayout 比较复杂, 因此我们先来看这个程序的运行结果, 再来讲解:

```

javac GridBagLayout.java
java GridBagLayout

```

⊕ 运行结果如图 3-11 所示:

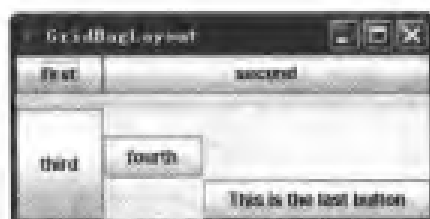


图 3-11

⊕ 说明:

- (1) 由于这是一个 Application 程序, 因此我们在第 15 行产生一个 JFrame 当作 Top-Level 组件。
- (2) 在第 19 行设置此 contentPane 的版面管理器为 GridBagLayout。
- (3) 在 22~32 行设置 GridBagConstraints 的参数。GridBagLayout 里的各种设置都是通过 GridBagConstraints 实现的, 而 GridBagConstraints 的参数共有 11 个, 我们慢慢地来看这 11 个参数的功用是什么, 等您看完这些参数的意义之后, 您会发现 GridBagLayout 并没有想象中的复杂。

gridx 与 gridy: 用来设置组件的位置。例如第 22~23 行均设置 `gridx=0, gridy=0`, 表示“first”这个按钮是放在第 0 行, 第 0 列的位置上, 而第 73~74 行均设置 `gridx=1, gridy=1`, 表示“fourth”这个按钮是放在第 1 行, 第 1 列的位置上。(您也可以使用 `GridBagConstraints.RELATIVE` 常数来设置组件的相关位置, 例如若将 `gridx` 设置为 `GridBagConstraints.RELATIVE`, 代表此组件位于之前所加入组件的右边。若将 `gridy` 设置为 `GridBagConstraints.RELATIVE`, 代表此组件位于以前所加入组件的下面。但笔者建议读者清楚地定义出 `gridx` 与 `gridy` 的位置, 在以后程序的维护上会比较轻松。)

gridwidth 与 gridheight: 用来设置组件所占的单位长度与高度, 默认值皆为 1。例如第 24~25 行, “first”按钮的 `gridwidth` 与 `gridheight` 均为 1, 表示“first”按钮长与宽各占一单位, 但第 41~42 行中, “second”按钮的 `gridwidth` 为 2 而 `gridheight` 为 1, 表示“second”按钮长为 2 单位, 但宽为 1 单位。如图 3-12 所示。(您可以使用 `GridBagConstraints.REMAINDER` 常量, 代表此组件为此行或此列的最后一个组件, 而且会占据所有剩余的空间。)

weightx 与 weighty: 用来设置当窗口变大时, 各组件跟着变大的比例, 数字越大, 表示组件能得到更多的空间, 默认值皆为 0。例如在第 26~27 行中, “first”按钮的 `weightx=10, weighty=1`, 而在第 43~44 行中, “second”按钮的 `weightx=1, weighty=1`, 这表示当窗口拉大时, “first”按钮变大的比例将大于“second”, 如图 3-12 所示(您也可以将 `weightx` 或 `weighty` 设置在 0.0~1.0 之间, 0.0 代表窗口变大时并不随着变大, 1.0 代表变大比率最大);



图 3-12

anchor: 当组件空间大于组件本身时, 要将组件置于何处, 有 CENTER (默认值)、NORTH、NORTHEAST、EAST、SOUTHEAST、SOUTH、SOUTHWEST、WEST、

NORTHWEST 可供选择。例如由于“third”这个按钮的高度设置得比较高，因此对同一列的组件来说，所得到的垂直空间也就比组件本身大多了，因此在第 96 行，我们设置 `anchor = GridBagConstraints.SOUTH`，使“This is the last button”按钮往下摆。同理，若我们将“fourth”按钮的 `anchor` 也设置为 `SOUTH`，所得到的图形将如图 3-13 所示。



图 3-13

`fill`: 当组件所处的位置有剩下的空间时，设置此参数可将剩余的空间填满，所提供的参数有 `NONE` (默认值)、`VERTICAL`、`HORIZONTAL`、`BOTH`。例如在上面的程序中，我们所设置的 `fill` 值均是 `HORIZONTAL`，因此当我们将窗口拉大时，会填满水平空间，但不会填满垂直空间，如图 3-14 所示。

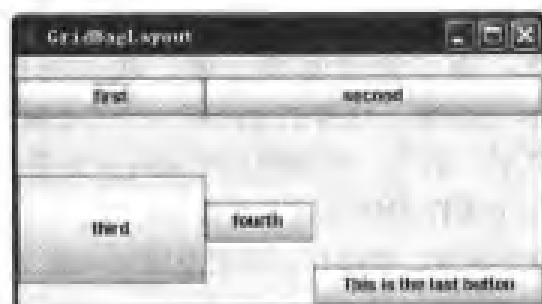


图 3-14

`insets`: 设置组件之间彼此的间距，它有四个参数，分别是上、左、下、右，默认值为 `(0,0,0,0)`。例如我们在第 64 行设置“third”按钮的间距为 `(10,0,0,0)`，因此“third”按钮与“first”按钮之间就有 10 单位的间距存在，若我们将此行间距改为 `(10,0,0,10)`，结果将如图 3-15 所示。



图 3-15

`ipadx` 与 `ipady`: 设置组件内的间距，默认值为 0。例如在第 65~66 行中，“third”按钮的 `ipadx` 为 0，`ipady` 为 50，因此就把“third”按钮给拉高了，在此你可以把

ipady 看成是“third”文字与本身按钮上下边缘之间的间距，如图 3-16 所示。

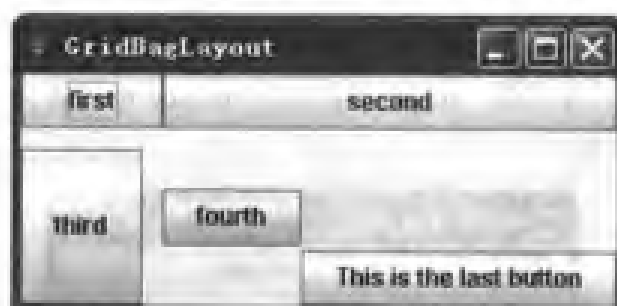


图 3-16

以上就是 GridBagConstraints 所有参数的介绍，虽然每个参数系统都有默认值存在，但笔者建议在利用 GridBagLayout 时，应清楚地定义每个参数的值，如此一来才会使程序容易维护与检测。

- (4) 我们以前提到过，GridBagLayout 里的各种设置都必须通过 GridBagConstraints，因此当我们将 GridBagConstraints 的参数都设置好之后，必须新建一个 GridBagConstraints 的对象出来，以便 GridBagLayout 使用，如第 33~34 行所示。而 GridBagLayout 要使用 GridBagConstraints 的对象必须使用 setConstraints 这个方法，如第 35 行。
- (5) 在上面的程序中，为了读者阅读方便，我们为每个组件均设置了 11 个 GridBagConstraints 的参数，这样的做法有一个好处就是使程序容易维护，不过程序看起来会比较冗长，若有一些参数每个组件所设置的值均相同，这时您可以只设置一次，以后每个组件均使用这个设置值，可使程序看起来比较精简。
- (6) 110~114 行是处理关闭窗口的操作，若您没写这一段，就算您已经关闭窗口了，但程序并不会终止。

3-2-7 BorderLayout 的使用

BoxLayout 的类层次结构图：

```
java.lang.Object
    --javax.swing.BoxLayout
```

BoxLayout 是 Swing 所提供的 Layout Manager，它可以提供多样化的版面管理方式，如同前面的 GridBagLayout 一样。跟 GridBagLayout 比较起来，BoxLayout 更容易使用，而且功能也非常强大。

BoxLayout 只有两种排列方式，一种是水平，一种是垂直，您可以使用 BoxLayout 所提供的两个常数 X_AXIS、Y_AXIS 来表示水平或垂直排列。另外有两点值得注意的是，当您使用水平的 BoxLayout 时，若摆放进去的组件不等高，则系统将会使所有的组件与最高组件等高，还有，若您将组件都摆在同一行时，系统不会因组件宽度大于 Container 的宽度，而使组件自动摆在下一行，您必须自行处理换行的操作。

表 3-7 是 BoxLayout 的构造函数：

表 3-7

构造函数
BoxLayout(Container target, int axis) 建立一个水平或垂直的 BoxLayout

讲到 BoxLayout, 我们就不得不提到 Box 这个 Container, Box 这个 Container 默认的 Layout 为 BoxLayout, 而它只能使用这个 Layout, 否则编译时会有 Error 产生, 如同前面所讲的, BoxLayout 是以水平或垂直方式排列, 因此, 当我们要产生一个 Box Container 时, 就必须指定它的排列方式, 表 3-8 是 Box 的构造函数:

表 3-8

构造函数
Box(int axis) 建立一个 Box Container, 并指定组件的排列方式是水平或垂直

上面的 axis 参数, 我们可以使用 BoxLayout.X_AXIS 或 BoxLayout.Y_AXIS 来指定。或是利用 Box 类所提供的两个方法: createHorizontalBox() 与 createVerticalBox() 来建立 Box Container。

Box 类提供 4 种透明的组件来做更方便的版面管理, 例如若我们有 A 与 B 两个连在一起的按钮, 如果我们在这两个按钮中间插入一个透明的组件, 则两个按钮看起来就具有分开的效果! 这就是透明组件的好处! Box 提供 4 种透明的组件, 分别是 Glue, Strut, Rigid, Filler, 我们下面将详细介绍这 4 种透明组件的差异。

我们以两个相连的按钮当例子, 起始状态如下:

范例 B1.java (文件位于随书光盘目录 exam\ch3\Box\B1.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class B1 {
6
7      public B1() {
8
9          JFrame f = new JFrame();
10
11          Container contentPane = f.getContentPane();
12          Box baseBox = Box.createHorizontalBox();
13          contentPane.add(baseBox);
14          baseBox.add(new JButton("A"));
15          baseBox.add(new JButton("B"));
16
17          f.setTitle("BoxLayout");
18          f.setSize(new Dimension(200,50));
19          f.setVisible(true);
20
21          f.addWindowListener(new WindowAdapter() {
22              public void windowClosing(WindowEvent e) {
23                  System.exit(0);

```

```

24         }
25     };
26 }
27
28 public static void main(String args[]) {
29
30     B1 b = new B1();
31
32 }
33 }

```

⊕ 说明：

- (1) 在程序第 12 行中，我们产生一个水平排列的 Box 组件，并在第 14~15 行中，将按钮 A 与 B 加入此 Box 组件中。
- (2) 第 18 行设置 Frame 的长宽各为 200 与 50 pixel。

⊕ 程序运行结果如图 3-17 所示。



图 3-17

Glue: 当 Glue 插入在两组件之间时，它会将两组件挤到最左与最右（或最上与最下），透明的 glue 将会占满整个中间的空间。

若我们在上面程序的第 14 与 15 行中间加入一行，如下所示：

范例 GlueB1.java (文件位于随书光盘目录 exam\ch3\Box\GlueB1.java)

```

14 baseBox.add(new JButton("A"));
15 baseBox.add(Box.createHorizontalGlue());
16 baseBox.add(new JButton("B"));

```

⊕ 说明：

在第 15 行中，我们加入一个水平透明的 Glue 组件，以便分开“A”与“B”这两个相连的按钮。

⊕ 程序运行结果如图 3-18 所示。



图 3-18

Strut: 当您不想将“A”与“B”按钮挤到最旁边时，您可以使用 Strut 组件，来设置所需要的大小，但仅能限定一维的大小，例如限定水平或垂直高度。

我们将上面程序做一点小小的更改：

范例 `StrutB1.java` (文件位于随书光盘目录 `exam\ch3\Box\StrutB1.java`)

```
14     baseBox.add(new JButton("A"));
15     baseBox.add(Box.createHorizontalStrut(50));
16     baseBox.add(new JButton("B"));
```

⊕ **说明:**

在第 15 行中, 删除原本的程序片段, 我们加入一个水平透明的 `Strut` 组件, 并设置长度为 50 pixels, 所以“A”与“B”按钮之间的间隔差距将只有 50 个像素, 而不会挤到最旁边了。

⊕ **程序运行结果如图 3-19 所示。**



图 3-19

Rigid: 这个透明组件跟 `Strut` 很像, 但它可以设置二维的限制, 也就是可以设置水平与垂直的限制宽度。我们直接来看一个例子, 就能清楚知道它们之间的差异了。

我们将上面的程序做一点小小的更改:

范例 `RigidB1.java` (文件位于随书光盘目录 `exam\ch3\Box\RigidB1.java`)

```
14     baseBox.add(new JButton("A"));
15     baseBox.add(Box.createRigidArea(new Dimension(50,50)));
16     baseBox.add(new JButton("B"));
17
18     f.setTitle("BoxLayout");
19     f.pack();
20     f.setVisible(true);
```

⊕ **说明:**

- (1) 在第 15 行中, 删除原本的程序片段, 我们加入一个 `Rigid` 组件, 并设置长宽各为 50 pixels, 所以“A”与“B”按钮之间的间隔差距有 50 个像素, 且高度也拉到 50 个像素。
- (2) 在第 19 行中, 我们为了将 `Rigid` 的高度功能显示出来, 因此以 `pack()` 方法代替 `setSize()` 方法。

⊕ **程序运行结果如图 3-20 所示。**

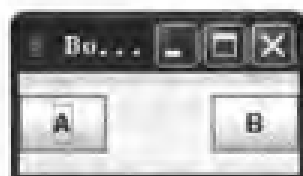


图 3-20

若我们将高度再拉长, 例如将 `baseBox.add(Box.createRigidArea(new Dimension(50,50)))` 的 `Dimension` 改成 `(50,100)`, 则两组件的上下空白将会增大, 因为透明的 `Rigid` 组件高度增高

了，运行结果如图 3-21 所示。



图 3-21

Filler: Filler 是 Box 的 inner class，它的功能跟 Rigid 很像，都可以指定长宽的大小限制，且 Filler 可以指定最大、较佳、最小的长宽大小。以下是 Filler 的构造函数：

表 3-9

构造函数

Box.Filler(Dimension min, Dimension pref, Dimension max)

建立一个指定大小的 Filler 对象

参数 min 表示最小的显示区域大小，如同上面的例子所示，若所设置最小区域的高度大于按钮“A”与“B”的高度，则按钮“A”与“B”的上方与下方将有空白出现。

pref 表示较佳的显示区域大小。max 表示最大的显示区域大小。

我们将上面程序修改成 Filler 的例子：

范例 FillerB1.java (文件位于随书光盘目录 exam\ch3\Box\FillerB1.java)

```
14 baseBox.add(new JButton("A"));
15 baseBox.add(new Box.Filler(new Dimension(50,50),
16                             new Dimension(100,50),
17                             new Dimension(200,50)));
18 baseBox.add(new JButton("B"));
```

⊕ 说明：

在第 15 行中，删除原本的程序片段，我们产生一个 Filler 组件，并设置它的最小值，较佳值，最大值。

⊕ 程序运行结果如图 3-22 所示。



图 3-22

由于我们是使用 HorizontalBox，因此我们将高度都设为 50，并以不同的 X 值（长度）来观察它的差异性。一开始系统会以较佳值，也就是“A”与“B”按钮的间距等于 100 像素，将画面显示出来，如上图所示。接下来用户可以去放大或缩小此画面，当您将此画面放大时，您会发现“A”与“B”按钮间的间距不会超过 200 像素，也就是我们所设置的最大值，如图 3-23 所示。

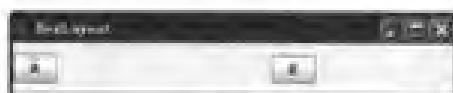


图 3-23

同样，当您将画面缩小时，您会发现“**A**”与“**B**”按钮间的间距不会小于 50 像素，也就是我们所设置的最小值，如图 3-24 所示。

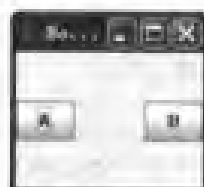


图 3-24

通过上述的介绍，相信现在您已经对 `BoxLayout` 不陌生了吧，由于 `Box` 所提供的方法常在 `BoxLayout` 中使用到，因此我们特别将它详细地列出来，如表 3-10 所示。

表 3-10

Box 类所提供的方法	
Static Component	<code>createGlue()</code> 构造一个 Glue 组件可向水平与垂直方向延伸
Static Box	<code>createHorizontalBox()</code> 构造一个水平排列的 Box 组件
Static Component	<code>createHorizontalGlue()</code> 构造一个水平的 Glue 组件
Static Component	<code>createHorizontalStrut(int width)</code> 构造一个水平的 Strut 组件
Static Component	<code>createRigidArea(Dimension d)</code> 构造一个 Rigid 组件
Static Box	<code>createVerticalBox()</code> 构造一个垂直排列的 Box 组件
Static Component	<code>createVerticalGlue()</code> 构造一个垂直的 Glue 组件
Static Component	<code>createVerticalStrut(int height)</code> 构造一个垂直的 Strut 组件
AccessibleContext	<code>getAccessibleContext()</code> 取得与 JComponent 相关连的 AccessibleContext
Void	<code>setLayout(LayoutManager l)</code> 丢出 <code>AWTError</code> ，因为 <code>Box</code> 只能使用 <code>BoxLayout</code>

趁热打铁，我们举个例子，利用 `BoxLayout` 与 `Box` 类来构造出一个复杂的组件排列方式，并熟悉它的运作方法。

范例 `BoxLayoutDemo.java` (文件位于随书光盘目录
 `exam\ch3\Box\BoxLayoutDemo.java`)

```
1   import java.awt.*;
2   import java.awt.event.*;
```



```
3    import javax.swing.*;
4
5    public class BoxLayoutDemo {
6
7        public BoxLayoutDemo() {
8
9            JFrame f = new JFrame();
10           Container contentPane = f.getContentPane();
11           Box baseBox = Box.createHorizontalBox();
12           contentPane.add(baseBox);
13
14           Box vBox = Box.createVerticalBox();
15           JButton b = new JButton("first");
16           vBox.add(b);
17           b = new JButton("third");
18           b.setMaximumSize(new Dimension(100,150));
19           vBox.add(b);
20           baseBox.add(vBox);
21
22           Box vBox1 = Box.createVerticalBox();
23           baseBox.add(vBox1);
24           b = new JButton("second");
25           b.setAlignmentX(Component.CENTER_ALIGNMENT);
26           b.setMaximumSize(new Dimension(300,50));
27           vBox1.add(b);
28
29           Box hBox = Box.createHorizontalBox();
30           vBox1.add(hBox);
31
32           Box vBox2 = Box.createVerticalBox();
33           vBox2.add(Box.createVerticalStrut(50));
34           vBox2.add(new JButton("fourth"));
35           vBox2.add(Box.createVerticalStrut(50));
36           hBox.add(vBox2);
37
38           Box vBox3 = Box.createVerticalBox();
39           vBox3.add(Box.createVerticalGlue());
40           vBox3.add(new JButton("This is the last button"));
41           hBox.add(vBox3);
42
43           f.setTitle("BoxLayout");
44           f.pack();
45           f.setVisible(true);
46
47           f.addWindowListener(new WindowAdapter() {
48               public void windowClosing(WindowEvent e) {
49                   System.exit(0);
50               }
51           });
52       }
53
54       public static void main(String args[]) {
55
56           BoxLayoutDemo b = new BoxLayoutDemo();
57       }
```

```

58     )
59     }
60

```

为了方便说明，我们先来看程序运行结果，再来解释程序代码，程序运行结果如图 3-25 所示。



图 3-25

说明：

上图是我们所要勾画出的组件摆设方式，因此我们在脑海中先要有一个初步的构想来设计如何摆设组件，我们可以先有一个水平排列方式的 Box 组件，用来当作基底容器，存放所有组件，接着依次将各个容器或组件加入此基底容器中，整个层次结构图如图 3-26 所示。

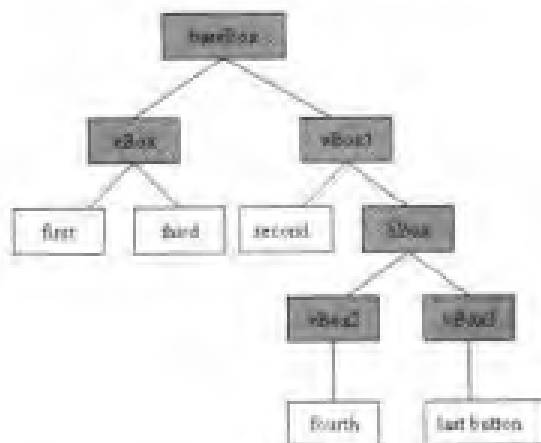


图 3-26

vBox 代表垂直排列的 Box 组件，也就是上图蓝色的部分，hBox 代表水平排列方式的 Box 组件，也就是黄色的部分，由上图您可以清楚地看出，我们总共用了 6 个 Box 组件来排列这 5 个按钮。

- (1) 在第 11 行中，我们先产生水平排列方式的 Box 组件，当作基底容器(BaseBox)。
- (2) 第 14~20 行，我们产生垂直排列方式的 Box 组件来安置第一与第三个按钮。
- (3) 第 22~30 行，我们产生垂直排列方式的 Box 组件来安置第二与另一个水平排列方式的 Box 组件。
- (4) 第 32~41 行，我们将第四与最后一个按钮加入水平排列方式的 Box 组件中。
- (5) 第 47~51 行是处理关闭窗口口的操作。

看完这个范例，您有没有发现这个图和以前的 `GridBagLayout` 有点像呢！而且程序代码干净许多，也比较容易看得懂，这说明了利用 `BoxLayout` 与 `Box` 组件可使版面的管理工作更为轻松与得意。而在这个范例中，我们全部都使用 `Box` 组件来放置按钮，其实您也可以利用其他版面管理方式，配合 `Box` 组件所提供的 `Glue` 或 `Strut` 等功能达到同样的效果，读者不妨可以试试看！

3-2-8 不使用版面管理器

一定要使用版面管理器来排列组件吗？当然不一定，您可以设置不要版面管理器，然后一一去设置每个组件的绝对位置与大小，一样可以如您所愿地将组件排列在您所希望的位置上，不过笔者并不建议这么做，因为当您使用版面管理器时，各个组件的大小会依窗口或字号等变动而跟着一起作适当地调整，若您没使用版面管理器，则因您是设置组件的绝对位置与大小，因此不管窗口或组件上的文字如何变动，组件的大小与位置依然不会改变。若不想使用版面管理器只需要在容器中设置版面管理方式为 `null` (`setLayout(null)`)，因为容器一般均有默认的版面管理方式，若不设置为 `null` 则将使用默认的版面管理器。我们举个简单的例子来看：

范例 `NonLayoutDemo.java` (文件位于随书光盘目录 `exam\ch3\NonLayoutDemo.java`)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class NonLayoutDemo {
6
7      public NonLayoutDemo() {
8
9          JFrame f = new JFrame();
10
11          Container contentPane = f.getContentPane();
12          contentPane.setLayout(null);
13          JButton b1 = new JButton("first");
14          contentPane.add(b1);
15          JButton b2 = new JButton("second");
16          contentPane.add(b2);
17
18          b1.setBounds(15, 10, 80, 30);
19          b2.setBounds(80, 50, 90, 40);
20
21          f.setTitle("NoLayout");
22          f.setSize(200,130);
23          f.setVisible(true);
24
25          f.addWindowListener(new WindowAdapter() {
26              public void windowClosing(WindowEvent e) {
27                  System.exit(0);
28              }
29          });
30      }
31  }
```

```
32     public static void main(String args[]) {  
33  
34         NonLayoutDemo b = new NonLayoutDemo();  
35  
36     }  
37 }  
38
```

⊕ 说明:

- (1) 第 12 行中, 我们设置 `contentPane` 的版面管理器为 `null`, 也就是没有版面管理器的意思。
- (2) 第 18~19 行, 我们设置按钮 “first” 与 “second” 的 X 与 Y 坐标, 宽度和高度。
- (3) 第 22 行中, 我们设置 `Frame` 的初始大小为 (200,130)。
- (4) 第 25~29 行是处理关闭窗口的操作。

编写好了之后, 我们进行编译与运行的操作:

```
javac NonLayoutDemo.java  
java NonLayoutDemo
```

我们可以看到如图 3-27 所示的运行结果:



图 3-27

若我们将窗口拉大, 可以发现上面两个按钮的位置与大小均不会有任何的改动, 如图 3-28 所示。

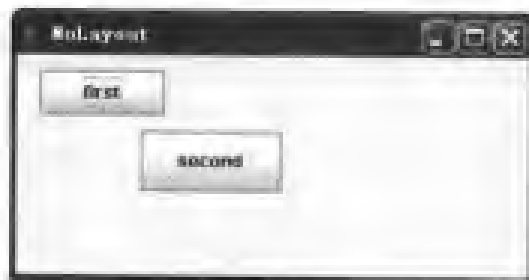


图 3-28

3-3 本章总结

在此章中我们介绍了各种版面管理器的功能与运用, 并针对各种细节加以讨论。若您的界面不需要十分花俏, 那么 `BorderLayout`、`FlowLayout` 与 `GridLayout` 的运用应已足够。若您需要多一点变化, 则可运用 `CardLayout`、`GridBagLayout` 与 `BoxLayout`, 而这些版面管理器可混杂地应用在同一界面上, 增加其变化性。另外, 有些 `Component`, 系统会给予管理器默认

值,例如 JPanel 的默认值为 FlowLayout, JRootPane 的 contentPane 为 BorderLayout 等等,不过笔者还是建议用户自己设置版面管理方式,以增加程序的可读性,并比较容易维护。

3-4 本章习题

- (1) 试述 JRootPane 所包含的层次结构,而这些功能各是什么?
- (2) 试述 Java 提供哪几种 Layout Manager,并练习各种 Layout Manager 的功能。
- (3) 在 3-2-4 节中的 GridLayoutDemo.java 中,若将第 12 行改成 `contentPane.setLayout(new GridLayout(0,3))`,运行时将会变成如图 3-29 所示,请解释其原因。



图 3-29

- (4) 试分别以 GridBagLayout 与 BoxLayout 画出如图 3-30 所示的图形。

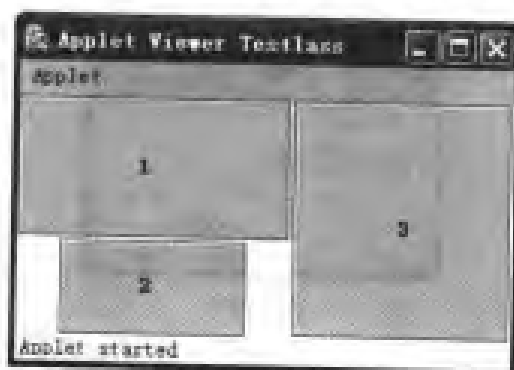


图 3-30

- (5) Box 组件能用 FlowLayout 吗?会产生什么错误?请解释其原因。

4

事件处理 (Event Handling)

当您辛辛苦苦地设计出良好的用户界面，却无法与用户交互时，就好比一辆好的跑车忘了加油，没什么实用的价值。事件的处理在用户界面部分占很大的重要性，如用户按下什么按钮，选了什么选项，是否按下键盘或鼠标等等事件，若能依软件的功能给予相对的反应，进而满足用户的需求，这样这套软件就好比有生命一般，可存活在计算机上，否则充其量又是成为硬盘中的垃圾，最后惨遭删除，真是惨不忍睹阿！

深入浅出

4-1 事件处理

相信大家都有安装软件的经验，在安装软件时常常会要求我们输入某些选项，例如选取某些包，然后按下一步等，当我们按下一步时，窗口怎么知道接下来要跳去哪里呢？这就是所谓的事件处理，也就是说，当用户激活了某一事件时，例如按下按钮、移动鼠标、按下鼠标等，系统会触发相对应的操作，例如运行某一个程序、跳出新的窗口、发出声音等等。

Java 自从 JDK1.1 之后，一改 JDK1.0 版以前的事件处理模式，目前都是以委托事件模式 (Delegation Event Model) 来处理用户所触发的事件。意思是说，当用户激活的事件不是我们所想要的事件时，我们就不需要加以处理，如此一来就可以增加系统的效能，而且整个事件的处理结构也会清楚许多。例如有一个窗口有两个按钮，一个按钮按下去会产生新的窗口，另一个按钮按下去会发出声音，当用户移动鼠标或敲击键盘时，均会产生事件 (Event)，但我们不需理会这些事件，因为这些事件对我们没有意义，我们只需要关心当用户按下按钮时所产生的事件即可。因此 Delegation Event Mode 就好像一部注册机，可以让用户注册所有想要捕获的事件。以打棒球为例，一个运动员只要关心对手的投球、教练的指示即可，而不用理会球迷的叫嚣、摄影机的镜头、甚至是空中的飞鸟（美国大联盟投手蓝迪强森曾在投球时砸死一只恰巧飞过的小鸟，不过这是个例外……）。至于什么事件我们要捕获下来，什么事件不用，在编写程序时必须事先规划好。

事件处理总共要注意 3 件事，第一是 Source 如何，也就是什么组件要被处理，如按钮、或是 CheckBox 等等。第二是什么样的事件要被处理，例如按下键盘或是按下按钮。若是按下按钮的事件要被处理，在按钮组件上就要增加 addActionListener 的事件处理类型，就如同刚刚讲的“注册”效果。第三是编写处理事件的程序代码，以按下按钮为例，处理这个事件的程序代码就必须写在 actionPerformed() 方法中。只要读者能清楚地知道这 3 件事，事件的处理将会是一件很轻松的事。

我们现在来了解一下 Swing 组件有哪些 Event Listener 可用！如同前面所述，我们必须先知道 Source 是什么，要驱动什么事件，最后才是编写处理事件的程序代码，因此我们将 Swing 组件作个整理，列出每个组件可能触发的 Event，以及相对应的 Event Listener，如表 4-1 所示。

表 4-1

Source	Event	Event Listener
AbstractButton (JButton, JToggleButton, JCheckBox, JRadioButton)	ActionEvent ChangeEvent ItemEvent	ActionListener ChangeListener ItemListener
JTextField JPasswordField	ActionEvent CaretEvent DocumentEvent UndoableEvent	ActionListener CaretListener DocumentListener UndoableListener
JTextArea	CaretEvent DocumentEvent UndoableEvent	CaretListener DocumentListener UndoableListener

续上表

Source	Event	Event Listener
JTextPane JEditorPane	CaretEvent DocumentEvent UndoableEvent HyperlinkEvent	CaretListener DocumentListener UndoableListener HyperlinkListener
JComboBox	ActionEvent ItemEvent	ActionListener ItemListener
JList	ListSelectionEvent ListDataEvent	ListSelectionListener ListDataListener
JFileChooser	ActionEvent	ActionListener
JMenuItem	ActionEvent ChangeEvent ItemEvent MenuKeyEvent MenuDragMouseEvent	ActionListener ChangeListener ItemListener MenuKeyListener MenuDragMouseListener
JMenu	MenuEvent	MenuListener
JPopupMenu	PopupMenuEvent	PopupMenuListener
JProgressBar	ChangeEvent	ChangeListener
JSlider	ChangeEvent	ChangeListener
JScrollBar	AdjustmentEvent	AdjustmentListener
JTable	ListSelectionEvent TableModelEvent TableColumnModelEvent CellEditorEvent	ListSelectionListener TableModelListener TableColumnModelListener CellEditorListener
JTabbedPane	ChangeEvent	ChangeListener
JTree	TreeSelectionEvent TreeExpansionEvent TreeWillExpandEvent TreeModeEvent	TreeSelectionListener TreeExpansionListener TreeWillExpandListener TreeModeListener
JTimer	ActionEvent	ActionListener

看到这个表，不知各位读者的心有没有凉了一半？其实不用担心，一般我们写程序所必须用到的 Event 通常种类不会很多，最常用的就是 ActionEvent、ItemEvent、ChangeEvent、WindowEvent 这几个事件，利用这几个事件处理方法，原则上可以处理大部分基本的需要，其他特殊事件的处理，通常是应用在特定的组件上，若有必要处理这些特殊事件，可再自行查表就行了。不过读者应该要对组件的事件有一个概念，也就是说，看到某个组件就联想一下这个组件差不多可能触发哪些事件，这样可以帮您在处理事件时更快速、方便。事实上，若您实际用过所有的 Swing 组件，您对组件可能触发的事件就不会这么陌生了。

1. 一般 ActionEvent 是发生在“按下按钮”、“选择了一个项目（如 Menu Item）”或是“在 JTextField 中按下【Enter】键”，这些均会产生 ActionEvent。
2. 一般 ItemEvent 是用在具有多个选项的组件上，例如 JCheckBox、JComboBox 或 JCheckBoxMenuItem 上。每个选项值为 on 或 off，当组件的状态由 on 变成



注意

off, 或由 off 变成 on, 我们就可以利用 ItemListener 中的 itemStateChanged- (ItemEvent e)方法, 作出相对应的操作, 如将某一个组件设为不可看见的 (Invisible)。

3. 一般 ChangeEvent 是用在可设定数值的拖曳杆或是可改变设定的选项上, 代表的是一种状态的改变, 例如 JSlider、JProgressBar 或 JTabbedPane。当用户拖曳 JSlider 上的滑动杆时, 滑动杆以前的设定值就会跟着改变, 此时便会产生 ChangeEvent, 我们就可以利用 ChangeListener 中的 stateChanged- (ChangeEvent e)方法处理相对应的操作。
4. 一般 WindowEvent 用于处理窗口的所有操作, 如处理 active 窗口或是 de-active 窗口, 处理窗口的最大、最小化, 处理窗口的关闭、打开、退出等。

以上除了 WindowEvent 外, 其余事件算是比较高层 (High-Level) 的事件, 因为只针对各个组件作处理。若我们要处理比较低层 (Low-Level) 的事件呢? 如鼠标、键盘的事件, 或是增加或删除一个组件等, 这些比较低层的事件, 我们就必需使用 AWT 所提供的一些事件处理方法。由下面的类层次结构表我们可以看出, 所有的 Swing 组件都是继承 java.awt.Component 类而来。

```
java.lang.Object
-- java.awt.Component
-- java.awt.Container
-- javax.swing.JComponent
-- javax.swing.各种 swing 组件
```

因此, 利用 java.awt.Component 与 java.awt.Container 类所提供的 Event Listener, 就可以让我们轻松处理低层的事件了, 这些低层的事件如表 4-2 所示。

表 4-2

Event	Listener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener
KeyEvent	KeyListener
MouseEvent	MouseListener
MouseMotionEvent	MouseMotionListener
WindowEvent	WindowListener

ComponentListener 主要处理组件大小的改变, 位置的改变, 或是可见与不可见状态 (hidden or visible) 等。

ContainerListener 主要处理组件的加入或移出容器 (Container)。

FocusListener 主要处理焦点的取得或移开焦点等操作。

MouseListener 主要就是处理鼠标是否在某个组件上, 是否按下鼠标键, 是否离开某个组件等操作。

MouseMotionListener 主要就是追踪鼠标的位置, 如 (X,Y) 坐标的位置。

WindowListener 主要处理窗口的所有操作, 如处理 active 窗口或是 de-active 窗口, 处理

窗口的最大、最小化，处理窗口的关闭、打开、退出等。

若读者写程序有查阅 Java API Document 的习惯（这是个很好的习惯，久而久之就可以慢慢了解 Java 的结构），可以发现上述的每种 `EventListener` 都是一种 `Interface`，里面只有定义这个 `EventListener` 的方法，因此当您要处理事件（Event）时，必须在类前面先 `Implements` 这个 `EventListener` 的 `Interface`，然后在这个类中，把要处理事件的程序代码写在此 `Interface` 的方法（Method）中。这是处理事件的标准操作，当然还有其他编写模式，例如利用 `Inner class` 的匿名类方法，就可以不用在 `class` 后面加上 `Implements EventListener` 的表示法，或是利用 `Adapter` 类，就不用一一将 `Interface` 中的每种方法都实现，此两种方法我们将在后面介绍。

让我们先来了解一下，每一种 `EventListener` 的 `Interface` 到底提供了哪几种方法要我们实现（Implement）。我们把事件区分成是由 `Swing` 或是 `AWT` 所引发两种，如表 4-3 和表 4-4 所示。

表 4-3

Swing EventListener	Method
<code>CaretListener</code>	<code>caretUpdate(CaretEvent e)</code>
<code>CellEditorListener</code>	<code>editingCanceled(ChangeEvent e)</code> <code>editingStopped(ChangeEvent e)</code>
<code>ChangeListener</code>	<code>stateChanged(ChangeEvent e)</code>
<code>DocumentListener</code>	<code>changedUpdate(DocumentEvent e)</code> <code>insertUpdate(DocumentEvent e)</code> <code>removeUpdate(DocumentEvent e)</code>
<code>HyperlinkListener</code>	<code>hyperlinkUpdate(HyperlinkEvent e)</code>
<code>ListDataListener</code>	<code>contentsChanged(ListDataEvent e)</code> <code>intervalAdded(ListDataEvent e)</code> <code>intervalRemoved(ListDataEvent e)</code>
<code>ListSelectionListener</code>	<code>valueChanged(ListSelectionEvent e)</code>
<code>MenuDragMouseListener</code>	<code>menuDragMouseDragged(MenuDragMouseEvent e)</code> <code>menuDragMouseEntered(MenuDragMouseEvent e)</code> <code>menuDragMouseExited(MenuDragMouseEvent e)</code> <code>menuDragMouseReleased(MenuDragMouseEvent e)</code>
<code>MenuKeyListener</code>	<code>menuKeyPressed(MenuKeyEvent e)</code> <code>menuKeyReleased(MenuKeyEvent e)</code> <code>menuKeyTyped(MenuKeyEvent e)</code>
<code>MenuListener</code>	<code>menuCanceled(MenuEvent e)</code> <code>menuDeselected(MenuEvent e)</code> <code>menuSelected(MenuEvent e)</code>
<code>PopupMenuListener</code>	<code>popupMenuCanceled(PopupMenuEvent e)</code> <code>popupMenuWillBecomeInvisible(PopupMenuEvent e)</code> <code>popupMenuWillBecomeVisible(PopupMenuEvent e)</code>
<code>TableColumnModelListener</code>	<code>columnAdded(TableColumnModelEvent e)</code> <code>columnMarginChanged(ChangeEvent e)</code> <code>columnMoved(TableColumnModelEvent e)</code> <code>columnRemoved(TableColumnModelEvent e)</code> <code>columnSelectionChanged(ListSelectionEvent e)</code>
<code>TableModelListener</code>	<code>tableChanged(TableModelEvent e)</code>

续上表

Swing EventListener	Method
TreeExpansionListener	treeCollapsed(TreeExpansionEvent event) treeExpanded(TreeExpansionEvent event)
TreeModelListener	treeNodesChanged(TreeModelEvent e) treeNodesInserted(TreeModelEvent e) treeNodesRemoved(TreeModelEvent e) treeStructureChanged(TreeModelEvent e)
TreeSelectionListener	valueChanged(TreeSelectionEvent e)
TreeWillExpandListener	treeWillCollapse(TreeExpansionEvent event) treeWillExpand(TreeExpansionEvent event)
UndoableEditListener	undoableEditHappened(UndoableEditEvent e)

表 4-4

AWT EventListener	Method
ActionListener	actionPerformed(ActionEvent e)
AdjustmentListener	adjustmentValueChanged(AdjustmentEvent e)
ComponentListener	componentHidden(ComponentEvent e) componentMoved(ComponentEvent e) componentResized(ComponentEvent e) componentShown(ComponentEvent e)
ContainerListener	componentAdded(ContainerEvent e) componentRemoved(ContainerEvent e)
FocusListener	focusGained(FocusEvent e) focusLost(FocusEvent e)
ItemListener	itemStateChanged(ItemEvent e)
KeyListener	keyPressed(KeyEvent e) keyReleased(KeyEvent e) keyTyped(KeyEvent e)
MouseListener	mouseClicked(MouseEvent e) mouseEntered(MouseEvent e) mouseExited(MouseEvent e) mousePressed(MouseEvent e) mouseReleased(MouseEvent e)
MouseMotionListener	mouseDragged(MouseEvent e) mouseMoved(MouseEvent e)
WindowListener	windowActivated(WindowEvent e) windowClosed(WindowEvent e) windowClosing(WindowEvent e) windowDeactivated(WindowEvent e) windowDeiconified(WindowEvent e) windowIconified(WindowEvent e) windowOpened(WindowEvent e)

以上我们列出了 Java 大部分的 `EventListener`，您可以针对不同的需求，使用不同的 `EventListener`，并实现 (Implements) `EventListener` 所提供的方法。接下来我们来看几个处理事件的例子，您就可以了解如何使用这些事件处理模式了。

4-2 事件处理范例说明

我们将在下面举几个例子来了解各种 High-Level 与 Low-Level 事件产生的时机与事件的处理方式。

4-2-1 `ActionEvent`、`WindowEvent` 与事件处理的多种写法

在下面这个例子，我们利用按钮产生 `ActionEvent`，并发出“哔”的音效。我们总共要处理按下按钮与关闭窗口的操作，因此在程序中必须实现 `ActionListener` 与 `WindowListener` 所定义的方法。

范例 `EventDemo1.java` (文件位于随书光盘目录 `exam\ch4\EventDemo1.java`)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class EventDemo1 implements ActionListener, WindowListener
6  {
7      public EventDemo1()
8      {
9          JFrame f = new JFrame("EventDemo1");
10         Container contentPane = f.getContentPane();
11         JButton b = new JButton("按我有声音喔");
12         b.addActionListener(this);
13         contentPane.add(b);
14         f.pack();
15         f.show();
16         f.addWindowListener(this);
17     }
18
19     public static void main(String args[])
20     {
21         new EventDemo1();
22     }
23
24     public void actionPerformed(ActionEvent e)
25     {
26         Toolkit.getDefaultToolkit().beep();
27     }
28
29     public void windowActivated(WindowEvent e){}
30     public void windowClosed(WindowEvent e){}
31     public void windowOpened(WindowEvent e){}
32     public void windowDeactivated(WindowEvent e){}
33     public void windowIconified(WindowEvent e){}
34     public void windowDeiconified(WindowEvent e){}
```

```

35     public void windowClosing(WindowEvent e)
36     {
37         System.exit(0);
38     }
39 }

```

◆ 说明：

- (1) 程序第 5 行，由于我们要运行按下按钮的操作与关闭窗口操作，因此我们需要处理两个事件，分别是 `ActionEvent` 与 `WindowEvent`，所以必须实现（implements）`ActionListener` 与 `WindowListener`。
- (2) 第 11~12 行，产生一个按钮，并使此按钮增加一个 `ActionListener`，使它具有捕获“按下按钮事件”的能力。`addActionListener(ActionListener l)`方法中，必须传入 `ActionListener` 对象当参数，而由于此 `EventDemo1` 类已经 implements `ActionListener` Interface，因此 `EventDemo1` 类也具有 `ActionListener` 的类型，所以我们可以将 `addActionListener()`方法中传入 `this` 参数，`this` 就是指 `EventDemo1` 类，若读者对这个保留字不熟悉，可参考一般的 Java 基础书籍。
- (3) 程序第 16 行，我们为了要处理关闭窗口的操作，因此在 `JFrame` 上增加一个 `WindowListener`，使它具有捕获“关闭窗口事件”的能力。
- (4) 程序第 24~27，由于 `ActionListener` 只有一个方法，那就是 `actionPerformed()`，因此我们将产生声音的效果写在这个方法中。
- (5) 程序第 29~38，由于 `WindowListener` 共有 7 个方法，我们必须全部实现这 7 个方法，但因为只关心关闭窗口的操作，因此只在 `windowClosing()`方法中写入关闭窗口的程序代码，其他方法均为空白。

◆ 图 4-1 为此程序的运行结果：



图 4-1

当您按下此按钮时，就会发出“哔”的一声，若您按下窗口右上角的“X”按钮，就可以将窗口关闭。

在上面的程序代码中，我们提到 `WindowListener` 共有 7 个方法，若我们只写 6 个呢？如我们只写了：

```

public void windowClosed(WindowEvent e){}
public void windowOpened(WindowEvent e){}
public void windowDeactivated(WindowEvent e){}
public void windowIconified(WindowEvent e){}
public void windowDeiconified(WindowEvent e){}
public void windowClosing(WindowEvent e)
{
    System.exit(0);
}

```

则在编译时, 就会有错误信息出现, 如图 4-2 所示。

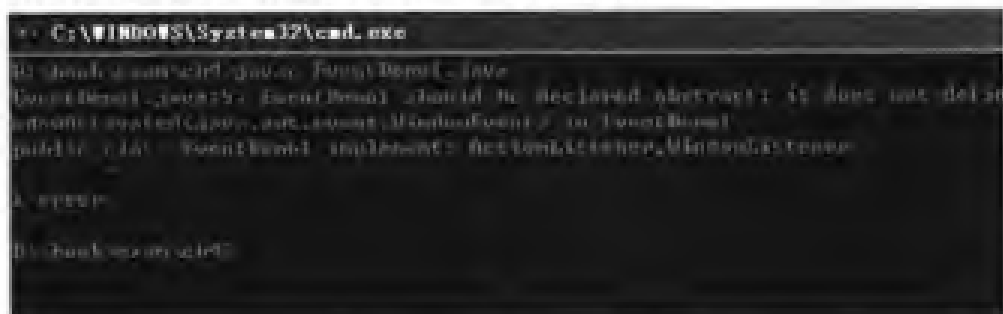


图 4-2

因此在写这样的程序时就要特别小心, 避免产生这样的错误。当然, 这样的写法感觉稍逊了一点, 动不动就要知道那个 `EventListener` 有哪几个方法, 而且还要全部写出来, 这在编写上非常容易出错。因此 Java 特别提供了另一种方法, 就是利用 `Adapter` 种类的类, 目的就是在使这些具有很多方法的 `EventListener` Interface, 集合成为一个抽象类。我们只需要继承这个抽象类, 然后覆写 (Override) 想要的方法即可, 就不需要在程序中将每个方法都写出来了。这样不仅可以避免程序写作上的错误, 也使程序看起来更清爽。一般我们常用的 `Adapter` 类如表 4-5 所示。

表 4-5

Adapter	对应的 EventListener
<code>java.awt.event.ComponentAdapter</code>	<code>ComponentListener</code>
<code>java.awt.event.ContainerAdapter</code>	<code>ContainerListener</code>
<code>java.awt.event.FocusAdapter</code>	<code>FocusListener</code>
<code>java.awt.event.KeyAdapter</code>	<code>KeyListener</code>
<code>java.awt.event.MouseAdapter</code>	<code>MouseListener</code>
<code>java.awt.event.MouseMotionAdapter</code>	<code>MouseMotionListener</code>
<code>java.awt.event.WindowAdapter</code>	<code>WindowListener</code>

我们利用 `Adapter` 的功能, 将上个程序改写如下:

范例 EventDemo2.java (文件位于随书光盘目录 exam\ch4\EventDemo2.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class EventDemo2 extends WindowAdapter implements ActionListener
6  {
7      public EventDemo2()
8      {
9          JFrame f = new JFrame("EventDemo2");
10         Container contentPane = f.getContentPane();
11         JButton b = new JButton("按我有声音喔");
12         b.addActionListener(this);
13         contentPane.add(b);
14         f.pack();
15         f.show();

```

```

16         f.addWindowListener(this);
17     }
18
19     public static void main(String args[])
20     {
21         new EventDemo2();
22     }
23
24     public void actionPerformed(ActionEvent e)
25     {
26         Toolkit.getDefaultToolkit().beep();
27     }
28
29     public void windowClosing(WindowEvent e)
30     {
31         System.exit(0);
32     }
33 }

```

说明

- (1) WindowAdapter 是一个抽象类，因此我们利用 Extends 的方式继承它。而 ActionListener 并没有相对应的 Adapter（这道理很简单，ActionListener 这个 Interface 只含有一个方法，利用 Adapter 的功能并不会带来什么好处，自然就不会有这种 Adapter 啰！），因此一样要用 implements 来实作 ActionListener 这个 Interface。
- (2) 程序第 29~32 行，由于我们已继承 WindowAdapter 抽象类，在此我们只关心窗口关闭的操作，因此只需写出这个方法的程序代码，其他方法我们就不需要再写出来了，相比较于上个程序，看起来是不是清爽多了呢。

程序运行结果将如同上个范例一样。

利用 Adapter 功能虽然很方便，但相对的问题也随之而来。由于 Adapter 种类的类均为抽象类，因此我们一定要用 Extends 来继承它。若今天我们要写的是一个 Applet 而不是 Application 呢？在程序一开始的地方我们就必须写成 Extends JApplet，而 Java 又不能多重继承，这时候要怎么运用 Adapter 种类的抽象类呢？还好，Java 在 1.1 版本之后就有所谓的 Inner Class 与 Inner Class 匿名类 (Anonymous Inner Classes)。利用这个机制我们就不需要在程序一开始的地方写 Extends xxxAdapter，这样就可以解决必须 Extends JApplet 又必须 Extends xxxAdapter 的多重继承问题了。

以下我们将上面的程序再改写，写成利用 Inner Class 匿名类的写法（事实上，这个写法我们在第 3 章已经用过很多次了）。

范例 EventDemo3.java (文件位于随书光盘目录 exam\ch4\EventDemo3.java)

```

1     import java.awt.*;
2     import java.awt.event.*;
3     import javax.swing.*;
4
5     public class EventDemo3
6     {

```

```

7      public EventDemo3()
8      {
9          JFrame f = new JFrame("EventDemo3");
10         Container contentPane = f.getContentPane();
11         JButton b = new JButton("按我有声音喔");
12         b.addActionListener(new ActionListener(){
13             public void actionPerformed(ActionEvent e)
14             {
15                 Toolkit.getDefaultToolkit().beep();
16             }
17         });
18         contentPane.add(b);
19         f.pack();
20         f.show();
21         f.addWindowListener(new WindowAdapter(){
22             public void windowClosing(WindowEvent e)
23             {
24                 System.exit(0);
25             }
26         });
27     }
28
29     public static void main(String args[])
30     {
31         new EventDemo3();
32     }

```

⊕ 说明:

- (1) 由于我们使用 Inner class 匿名类的方式, 因此在程序第 5 行中, 就不需要写 Extends 或 Implements 这样的语句。
- (2) 程序第 12 行, addActionListener(ActionEvent e)要传一个 ActionEvent 的对象进去, 在此我们利用 Inner class 匿名类的特性, 直接 new 一个 ActionListener 对象, 并实作 (Implements) actionPerformed()方法。当用户按下按钮时, ActionListener 就会得到一个 ActionEvent 对象, 获知用户已经按下按钮, 然后程序就会跟着去运行 actionPerformed()中的程序代码。
- (3) 程序第 20 行, 跟第 12 行差不多, addWindowListener(ActionEvent e)要传一个 WindowEvent 的对象进去, 我们利用 Inner class 匿名类与 Adapter 的特性, 直接 new 一个 WindowAdapter 对象。由于是使用 Adapter 抽象类, 因此只需实作 (Implements) windowClosing()方法。当用户按下关闭窗口的“X”按钮时, WindowAdapter 就会得到一个 windowEvent 对象, 知道用户欲关闭窗口, 因此程序就会跟着去运行 windowClosing 中的程序代码。

若读者以后有机会看别人所写的程序代码, 可发现事件处理模式几乎都以这三种方式来表达, 不过, 当然不只有这几种写法, 您可以有很多变化, 但主要就是必须将 addXXXListener()方法传入 XXXEvent 里去, 而如何获得这个 XXXEvent, 就看您要怎么做了。如在 class 上 Implements XXXListener 或是 Extends XXXAdapter, 或是利用 Inner class 匿名类。以下我们再对这个例子作一个变化。下面这个例子 EventDemo4.java 包含两个 class, EventDemo4 class 负责加入组件与显示窗口, EventHandle class 负责事件处理部分, 这样的结构可以让人很清

楚地知道到底要处理什么事件，处理方式是什么？而 EventDemo4 class 的程序代码只需专研它所处理的事，而不用在乎事件要如何处理。不过此结构有一个缺点，那就是当系统很大、有很多组件（Component）都必须作事件处理时，这样的写法有时看起来反而不容易阅读。所以，在设计事件处理模式时，读者可依整个程序的概念结构，选择比较适合的事件处理模式。

范例 EventDemo4.java (文件位于随书光盘目录 exam\ch4\EventDemo4.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class EventDemo4
6  {
7      public EventDemo4()
8      {
9          JFrame f = new JFrame("EventDemo4");
10         Container contentPane = f.getContentPane();
11         JButton b = new JButton("按我有声音喔");
12         b.addActionListener(new EventHandle());
13         contentPane.add(b);
14         f.pack();
15         f.show();
16         f.addWindowListener(new EventHandle());
17     }
18
19     public static void main(String args[])
20     {
21         new EventDemo4();
22     }
23 }
24
25 class EventHandle extends WindowAdapter implements ActionListener
26 {
27     public void actionPerformed(ActionEvent e)
28     {
29         Toolkit.getDefaultToolkit().beep();
30     }
31
32     public void windowClosing(WindowEvent e)
33     {
34         System.exit(0);
35     }
36 }

```

⊕ 说明：

- (1) 程序第 25 行，我们建立一个专门处理事件的类：EventHandle class，这个类继承了 WindowAdapter，并实作了 ActionListener。我们将此类放在 EventDemo4 类下面是为了说明方便，事实上读者可将此类存成另外一个 Java 文件！
- (2) 程序第 12 行，addActionListener(ActionEvent e) 要传一个 ActionEvent 的对象进去，由于 EventHandle 类实作了 ActionListener 类，因此它便具有处理 ActionEvent 的能力，所以我们 new 一个 EventHandle 对象当作是 addActionListener 参数。当用户按

下按钮时, `EventHandle` 就会得到一个 `ActionEvent` 事件, 并寻找 `actionPerformed()` 方法来处理该事件。

(3) 程序第 16 行如同程序第 12 行一般。

4-2-2 相同组件事件的处理

上面我们处理事件的情形, 都是在同一个组件上动手脚。若今天我们有二个按钮都要处理按下的操作 (`ActionEvent`), 并运行不同的程序, 那要怎么做呢? 在 Java 中提供两种方法让您知道到底是哪个组件触发了事件, 那就是 `getSource()` 与 `getActionCommand()`, `getActionCommand()` 方法是 `ActionEvent` 类所提供, 而 `getSource()` 方法是 `EventObject` 类所提供, 但 `ActionEvent` 类继承了 `EventObject` 类, 因此这两种方法 `ActionEvent` 都可以使用。我们来看下面的例子, 您就能知道如何使用这两种方法了。

范例

`EventDemo5.java` (文件位于随书光盘目录 `exam.ch4\EventDemo5.java`)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class EventDemo5 extends WindowAdapter implements ActionListener
6  {
7      JButton b1 = null;
8      JButton b2 = null;
9
10     public EventDemo5()
11     {
12         JFrame f = new JFrame("EventDemo5");
13         Container contentPane = f.getContentPane();
14         contentPane.setLayout(new GridLayout(1,2));
15         b1 = new JButton("按我有声音喔");
16         b2 = new JButton("按我可开新窗口");
17         b1.addActionListener(this);
18         b2.addActionListener(this);
19         contentPane.add(b1);
20         contentPane.add(b2);
21         f.pack();
22         f.show();
23         f.addWindowListener(this);
24     }
25
26     public void actionPerformed(ActionEvent e)
27     {
28         if(e.getSource()==b1)
29             Toolkit.getDefaultToolkit().beep();
30         if(e.getSource()==b2)
31         {
32             JFrame newF = new JFrame("新窗口");
33             newF.setSize(200,200);
34             newF.show();
35         }
36     }

```

```
37
38     public void windowClosing(WindowEvent e)
39     {
40         System.exit(0);
41     }
42
43     public static void main(String args[])
44     {
45         new EventDemo5();
46     }
47 }
48
```

说明：

- (1) 程序第 5 行, EventDemo5 class 继承 WindowAdapter 并实作 ActionListener Interface。
- (2) 程序第 14 行, 我们将 JFrame 的 contentPane 设为一行两列的 GridLayout。
- (3) 程序第 19~20 行, 将按钮 b1, b2 加入 contentPane 中。
- (4) 程序第 26~36 行, 实作 ActionListener 中所提供的 actionPerformed() 方法。
- (5) 程序第 28 与 30 行, 由 getSource() 方法判断到底是哪个按钮被按下, 并运行相关的操作, 如果是按下 b1 按钮, 则发出“哔”一声, 如果按下 b2 按钮, 则开新窗口。
- (6) 程序第 38 行, 由于我们继承 WindowAdapter 抽象类, 在此我们只是要处理窗口关闭的操作, 因此只需编写 windowClosing() 方法即可!

程序运行结果如图 4-3 所示。



图 4-3

当您按下左边的按钮时, 就会有“哔”一声, 当您按下右边的按钮时, 就会产生如图 4-4 所示的新窗口:



图 4-4

接下来, 我们将 EventDemo5.java 改换成用 getActionCommand() 方法来判断是哪个按钮产生的事件。

范例 EventDemo6.java (文件位于随书光盘目录 exam\ch4\EventDemo6.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class EventDemo6 extends WindowAdapter implements ActionListener
6  {
7      JButton b1 = null;
8      JButton b2 = null;
9
10     public EventDemo6()
11     {
12         JFrame f = new JFrame("EventDemo6");
13         Container contentPane = f.getContentPane();
14         contentPane.setLayout(new GridLayout(1,2));
15         b1 = new JButton("按我有声音喔");
16         b2 = new JButton("按我可开新窗口");
17         b1.addActionListener(this);
18         b2.addActionListener(this);
19         contentPane.add(b1);
20         contentPane.add(b2);
21         f.pack();
22         f.show();
23         f.addWindowListener(this);
24     }
25
26     public void actionPerformed(ActionEvent e)
27     {
28         if((e.getActionCommand()).equals("按我有声音喔"))
29             Toolkit.getDefaultToolkit().beep();
30         if((e.getActionCommand()).equals("按我可开新窗口"))
31         {
32             JFrame newF = new JFrame("新窗口");
33             newF.setSize(200,200);
34             newF.show();
35         }
36     }
37
38     public void windowClosing(WindowEvent e)
39     {
40         System.exit(0);
41     }
42
43     public static void main(String args[])
44     {
45         new EventDemo6();
46     }
47 }

```

◆ 说明:

- (1) 程序第 28 与 30 行, 利用 `getActionCommand()` 方法会返回按钮上的文字字符串, 我们利用这个字符串来比对到底是哪个按钮发生事件, 并运行相关的操作, 如按

下 b1 按钮，则发出“哔”一声，按下 b2 按钮，则开新窗口。

程序运行结果如上例。

4-2-3 鼠标事件处理

刚刚我们看的都是针对某一个组件作事件处理，是属于“High-Level”的事件。而对于如控制鼠标这种“Low-Level”的事件，我们将在下面作详细的说明。

什么时候会用到鼠标事件呢？其实使用到的场合相当多，例如将鼠标移到一个文件上，按一下可让此文件获得焦点（Focus），按两下可运行此文件。或是文件之间的拖曳即开两个窗口，拉一个文件从原来窗口到另一个窗口，以便进行文件的复制，此时您就必须判断放开鼠标的位置是否在另一个窗口上。若是，才进行文件复制。或者，您可以在任何游戏中发现检测鼠标位置的重要性。如一般的战略游戏，您必须移动游戏中的人物到适当位置，然后再采取其他操作。当然应用不只这些，不过我们可以看出，处理鼠标事件是经常遇见的事，也非常的实用。

在上节我们有提到，与 Mouse 有关的事件可分成两类，一种是 MouseListener Interface，共提供 5 种方法，主要是针对鼠标按键与位置作检测。一种是 MouseMotionListener Interface，共提供 2 种方法，主要是针对鼠标坐标与拖曳操作作处理，如表 4-6 所示。

表 4-6

MouseListener	mouseClicked(MouseEvent e) mouseEntered(MouseEvent e) mouseExited(MouseEvent e) mousePressed(MouseEvent e) mouseReleased(MouseEvent e)
MouseMotionListener	mouseDragged(MouseEvent e) mouseMoved(MouseEvent e)

上面这两个 Listener Interface，Java 都有提供相对应的 Adapter 抽象类，方便用户使用，分别是 MouseAdpater 与 MouseMotionAdapter。以下我们就来看如何处理 Mouse 事件吧！

在下面这个范例中我们可以在一个 JFrame 中放入一个 JLabel 与 JButton，我们去检测鼠标光标是否在 JButton 上，是否按下按钮，是否放开按钮等，并将结果显示在 JLabel 上。

范例 MouseDemo1.java (文件位于随书光盘目录 exam\ch4\MouseDemo1.java)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class MouseDemo1 extends WindowAdapter implements MouseListener
6  {
7      JFrame f = null;
8      JButton b1 = null;
9      JLabel label = null;
10
11     public MouseDemo1()
12     {
13         f = new JFrame("MouseDemo1");
```

```
14         Container contentPane = f.getContentPane();
15         contentPane.setLayout(new GridLayout(2,1));
16         b1 = new JButton("按钮");
17         label = new JLabel("起始状态, 还没鼠标事件", JLabel.CENTER);
18         b1.addMouseListener(this);
19         contentPane.add(label);
20         contentPane.add(b1);
21         f.pack();
22         f.show();
23         f.addWindowListener(this);
24     }
25
26     public void mousePressed(MouseEvent e) {
27         label.setText("您已经压下鼠标按钮");
28     }
29
30     public void mouseReleased(MouseEvent e) {
31         label.setText("您已经放开鼠标按钮");
32     }
33
34     public void mouseEntered(MouseEvent e) {
35         label.setText("鼠标光标进入按钮");
36     }
37
38     public void mouseExited(MouseEvent e) {
39         label.setText("鼠标光标离开按钮");
40     }
41
42     public void mouseClicked(MouseEvent e) {
43         label.setText("您已经按下鼠标按钮");
44     }
45
46     public void windowClosing(WindowEvent e)
47     {
48         System.exit(0);
49     }
50
51     public static void main(String args[])
52     {
53         new MouseDemo1();
54     }
55 }
```

◆ 说明:

- (1) 程序第 5 行, 我们继承了 WindowAdapter 抽象类, 并实作 MouseListener Interface, 因此我们必须把 MouseListener 中的 5 种方法都实现。
- (2) 程序第 15 行, 我们将 contentPane 设为两行一列的 GridLayout, 将 JLabel 放在上面, 而 JButton 放在下面。
- (3) 程序第 18 行, 我们将按钮 b1 加入 MouseListener 中, 来检测按钮上所有的鼠标操作。

(4) 程序第 26~44 行, 实作 `MouseListener` 中的 5 种方法, 在此我们只是把状态显示在 `JLabel` 上而已!

✎ 运行结果如下:

一开始, 当鼠标光标还没经过按钮时, 上面的文字会显示“起始状态, 还没鼠标事件”, 如图 4-5 所示。



图 4-5

当鼠标光标移到按钮上时, 上面的文字会显示“鼠标光标进入按钮”, 如图 4-6 所示。



图 4-6

当您压下按钮时, 上面文字会显示“您已经压下鼠标按钮”, 如图 4-7 所示。



图 4-7

读者可自行试试这个简单的程序。

如果我们想知道 `Mouse` 点选的位置在哪, 或是想知道用户连续按了几次鼠标, 我们可以使用 `MouseEvent` 类所提供的 `getX()`, `getY()` 与 `getClickCount()` 方法。下面这个范例中, 我们可以知道当在按钮上按下鼠标时的坐标位置与连续按下两次按钮后就会产生一个新窗口。

范例 `MouseDemo2.java` (文件位于随书光盘目录 `exam\ch4\MouseDemo2.java`)

```
1 import java.awt.*;  
2 import java.awt.event.*;  
3 import javax.swing.*;  
4  
5 public class MouseDemo2 extends MouseAdapter  
6 {  
7     JFrame f = null;
```

```

8      JButton b1 = null;
9      JLabel label = null;
10
11     public MouseDemo2()
12     {
13         f = new JFrame("MouseDemo2");
14         Container contentPane = f.getContentPane();
15         contentPane.setLayout(new GridLayout(2,1));
16         b1 = new JButton("按钮");
17         label = new JLabel("起始状态, 还没鼠标事件", JLabel.CENTER);
18         b1.addMouseListener(this);
19         contentPane.add(label);
20         contentPane.add(b1);
21         f.pack();
22         f.show();
23         f.addWindowListener(new WindowAdapter() {
24             public void windowClosing(WindowEvent e)
25             {
26                 System.exit(0);
27             }
28         });
29
30         public void mousePressed(MouseEvent e) {
31             label.setText("目前鼠标坐标 (X,Y) 为: (" + e.getX() + ", " + e.getY() + ")");
32         }
33
34         public void mouseClicked(MouseEvent e) {
35             if (e.getClickCount() == 2)
36             {
37                 JFrame newF = new JFrame("新窗口");
38                 newF.setSize(200,200);
39                 newF.show();
40             }
41         }
42
43         public static void main(String args[])
44         {
45             new MouseDemo2();
46         }
47     }

```

⊕ 说明:

- (1) 在程序第 5 行中, 我们直接继承 MouseAdapter 抽象类。由于 Java 不能多重继承, 因此 MouseDemo2 类不能再 extends WindowAdapter, 在此改以 Inner class 匿名类方式处理关闭窗口操作。
- (2) 程序第 31 行, 当用户在按钮上压下鼠标键时, 就会在按钮上面显示目前的鼠标坐标。
- (3) 程序第 35 行, 当用户在按钮上连续按两下时, 就会产生一个新窗口。

⊕ 程序运行结果如下:

当用户在按钮上压下鼠标键时, 会显示出目前的鼠标坐标 (X,Y), 如图 4-8 所示。

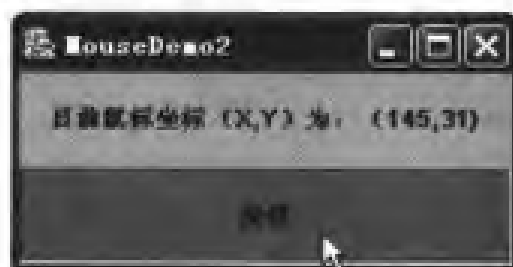


图 4-8

当用户连续按两次按钮时，即可产生如图 4-9 所示的窗口。



图 4-9

接下来我们来讨论 `MouseMotionListener` 的使用时机。这个 `Interface` 共定义了 `mouseMoved()` 与 `mouseDragged()` 两种方法，可让你随时掌握鼠标的坐标，并处理拖曳鼠标的操作。下面这个范例让您知道鼠标在 `JFrame` 上的坐标，并拖曳鼠标画出直线出来。

范例 `MouseDemo3.java` (文件位于随书光盘目录 `exam\ch4\MouseDemo3.java`)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class MouseDemo3 extends JFrame implements MouseListener, Mouse
6  MotionListener
7  {
8      int flag ; // flag =1 代表 Mouse Moved, flag =2 代表 Mouse Dragged
9      int x = 0; //坐标变量
10     int y = 0; //坐标变量
11     int startx, starty, endx, endy; //起始坐标与终点坐标
12
13     public MouseDemo3()
14     {
15         Container contentPane = getContentPane();
16         contentPane.addMouseListener(this);
17         contentPane.addMouseMotionListener(this);
18         setSize(300,300);
19         show();
20         addWindowListener(new WindowAdapter(){
21             public void windowClosing(WindowEvent e)
22             {
23                 System.exit(0);
24             }
25         });
26     }
27 }

```

```
25     }
26
27     public void mousePressed(MouseEvent e) {
28         startx = e.getX();
29         starty = e.getY();
30     }
31
32     public void mouseReleased(MouseEvent e) {
33         endx = e.getX();
34         endy = e.getY();
35     }
36
37     public void mouseEntered(MouseEvent e) {
38     }
39
40     public void mouseExited(MouseEvent e) {
41     }
42
43     public void mouseClicked(MouseEvent e) {
44     }
45
46     public void mouseMoved(MouseEvent e) {
47         flag = 1;
48         x = e.getX();
49         y = e.getY();
50         repaint();
51     }
52
53     public void mouseDragged(MouseEvent e) {
54         flag = 2;
55         x = e.getX();
56         y = e.getY();
57         repaint();
58     }
59
60     public void update(Graphics g)
61     {
62         g.setColor(this.getBackground());
63         g.fillRect(0,0,getWidth(),getHeight());
64         paint(g);
65     }
66     public void paint(Graphics g)
67     {
68         g.setColor(Color.black);
69         if (flag == 1)
70         {
71             g.drawString("鼠标坐标: (" + x + ", " + y + ")", 10, 50);
72             g.drawLine(startx, starty, endx, endy);
73         }
74         if (flag == 2)
75         {
76             g.drawString("拖曳鼠标坐标: (" + x + ", " + y + ")", 10, 50);
77             g.drawLine(startx, starty, x, y);
78         }
79     }
```

```
80
81     public static void main(String args[])
82     {
83         new MouseDemo3();
84     }
85 }
```

⊕ 说明：

- (1) 这个程序的写法与以前有稍稍不同，一开始在程序第 5 行中，我们先继承 JFrame 类，使 MouseDemo3 具有 JFrame 的性质，并且为了达到画线的功能，我们分别 implements MouseListener 与 MouseMotionListener。
- (2) 程序第 16~17 行，增加 MouseListener 与 MouseMotionListener 到 contentPane 中。
- (3) 程序第 27~35 行，由方法 mousePressed() 与 mouseReleased() 取得鼠标拖曳的开始与结束坐标。
- (4) 程序第 46~58 行，由方法 mouseMoved() 与 mouseDragged() 取得鼠标移动的每一个坐标，并调用 repaint() 方法。
- (5) repaint() 方法被调用后，会去调用 update() 方法，再由 update() 方法调用 paint() 方法，update() 与 paint() 的参数 Graphics 会自动由系统传入，用户不要担心。在此，我们将覆写 (Override) update() 方法，目的是要让以前的画面清除掉，只保留最新的画面。
- (6) 程序第 66~79 行，编写 paint() 方法，当 mouse 坐标移动时，会输出移动时的各个坐标。当 mouse 拖曳移动时，则不仅输出移动坐标，且在放开鼠标后，会在 JFrame 上画出拖曳路径的一条直线。

⊕ 程序运行结果如下：

当鼠标进入 JFrame 窗口时，坐标会随着鼠标的移动而改变，如图 4-10 所示。



图 4-10

当拖曳鼠标时，则输出鼠标坐标与拖曳路径的一条直线，如图 4-11 所示。

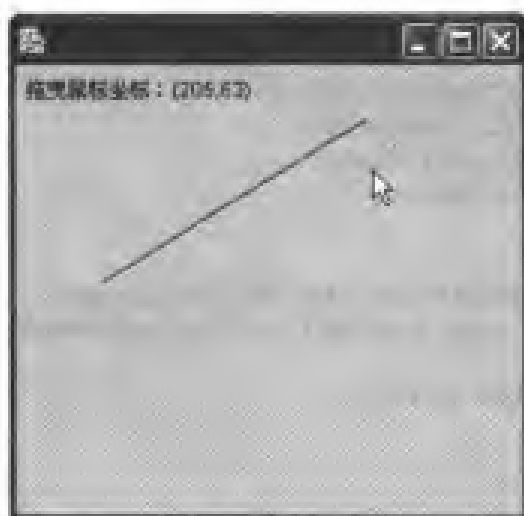


图 4-11

4-2-4 键盘事件处理

处理完鼠标事件之后，我们再来讨论键盘事件，毕竟这两项是我们最常用的输入设备。我们利用 `KeyListener` 这个 `Interface` 来处理有关键盘的事件。这个 `Interface` 共定义了 3 种方法，分别是 `keyPressed()`、`keyTyped()` 与 `keyReleased()`。其实跟 `MouseListener` 所提供的方法有点像，只是 `MouseListener` `Interface` 多了 `mouseEntered()` 与 `mouseExited()` 罢了。`KeyListener` `Interface` 也有相对应的 `KeyAdapter` 抽象类可以使用，方便用户编写程序。

我们直接举个例子让读者了解如何处理键盘事件。

在这个例子中，我们在 `JFrame` 上放进 `JLabel`、`JTextField` 与 `JButton` 这三个组件。当用户打字到 `JTextField` 时，在 `JLabel` 上也会跟着显示出来。若用户按【O】键，则会打开一个新窗口。当用户按下 `JButton` 时，会清除 `JLabel` 与 `JTextField` 上的数据。

范例 KeyDemo.java (文件位于随书光盘目录 exam\ch4\KeyDemo.java)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class KeyDemo extends KeyAdapter implements ActionListener
6  {
7      JFrame f = null;
8      JLabel label = null;
9      JTextField tField = null;
10     String keyString = "";
11
12     public KeyDemo()
13     {
14         f = new JFrame("KeyEventDemo");
15         Container contentPane = f.getContentPane();
16         contentPane.setLayout(new GridLayout(3,1));
17         label = new JLabel();
18         tField = new JTextField();
19         tField.requestFocus();
```

```

20         tField.addKeyListener(this);
21         JButton b = new JButton("清除");
22         b.addActionListener(this);
23         contentPane.add(label);
24         contentPane.add(tField);
25         contentPane.add(b);
26         f.pack();
27         f.show();
28         f.addWindowListener(new WindowAdapter() {
29             public void windowClosing(WindowEvent e)
30             {
31                 System.exit(0);
32             }
33         });
34
35         public void actionPerformed(ActionEvent e)
36         {
37             keyString = "";
38             label.setText("");
39             tField.setText("");
40             tField.requestFocus();
41         }
42
43         public void keyTyped(KeyEvent e)
44         {
45             char c = e.getKeyChar();
46             if(c=='o')
47             {
48                 JFrame newF = new JFrame("新窗口");
49                 newF.setSize(200,200);
50                 newF.show();
51             }
52             keyString = keyString + c;
53             label.setText(keyString);
54         }
55
56         public static void main(String args[])
57         {
58             new KeyDemo();
59         }
60     }

```

◆ 说明:

- (1) 程序第 5 行中, 我们继承 KeyAdapter 抽象类, 并 implements ActionListener Interface。
- (2) 程序第 19 行, 我们使光标一开始就放在 tField 上, 以方便输入。
- (3) 程序第 20 行, 增加 KeyListener 到 tField 中。
- (4) 程序第 35~41 行, 当按下按钮时运行此清除 tField 与 label 上文字的工作。
- (5) 程序第 43~54 行, 判断用户输入的字符, 若为“O”, 则打开新窗口, 且将用户输入的字符在 label 中输出。

◆ 程序运行结果如图 4-12 所示。



图 4-12

输入英文字母“O”之后，会产生一个新窗口，如图 4-13 和图 4-14 所示。



图 4-13



图 4-14

除了上面所提的 `getKeyChar()` 方法外，`KeyEvent` 类还有两种方法也常常被用到，那就是 `getKeyCode()` 与 `getKeyModifiersText(int modifiers)`。键盘上每一个按钮都有对应码 (Code)，可用来查知用户按了什么键，如【Shift】键 Code 为 16。利用 `getKeyCode()` 方法就可以得知这个码，不过读者要注意，这个方法在 `keyTyped()` 上是无法检测出来的，因为 `keyTyped()` 只管用户输入的字符，而不会管到键盘的对应码，算是处理比较高层 (High-Level) 事件的方法。也就是说 `keyTyped()` 方法是 keyboard independent，因为不同的键盘可能有不同的对应码 (如 Windows U.S. keyboard 与 Windows French keyboard 就有不同的对应码)。因此您一定要将 `getKeyCode()` 方法写在 `keyPressed()` 或 `keyReleased()` 方法中才会有效，因为这两个方法是处理比较低层 (Low-Level) 的方法。

另外 `getKeyModifiersText()` 方法可返回修饰键的字符串，如返回 “Shift” 字符串或是 “Ctrl+Shift” 字符串，不过你要先传入 `modifiers` 参数。您可以直接使用 `getModifiers()` 方法来得到 `modifiers` 参数。这个方法是定义在 `InputEvent` 类中，而 `KeyEvent` 继承它，因此就能直接使用这个方法。同样，您必须将 `getKeyModifiersText()` 与 `getModifiers()` 方法放在 `keyPressed()` 或 `keyReleased()` 方法中才会有效。我们将此练习放在习题中，读者不妨试试看。

4-3 本章总结

在本章中，我们说明了事件处理的 3 步骤：第一是 Source 是什么；第二是什么样的事件要被处理；第三是编写处理事件的程序代码。在说明事件处理步骤后，我们列出了大部分的 `EventListener Interface`，并列出的方法。而在后半段，我们以实际例子说明如何编写处理

事件的方法，并介绍 4 种不同的编写模式。最后，我们以鼠标与键盘事件当结尾，让读者体验到事件处理的重要性，并轻松处理事件的驱动。在往后的章节中，我们还会在适当的时机，针对不同的 Swing 组件处理不同的事件来源，读者可发现事件处理在窗口设计中占了非常重要的角色！

4-4 本章习题

1. 试说明事件处理的三步骤。
2. 继承 Adapter 类有什么优缺点！
3. 试练习 Event Handling 的多种写法，如 extends XXXAdapter、或是 implements XXXListener、或是利用 Inner class 的匿名类、或是混合使用。
4. 若有两个按钮，如何分辨是哪个按钮驱动了事件呢？
5. 试利用 KeyEvent 中的 getKeyCode()方法得到相关的键盘对应码；若有按到修饰键，如【Shift】键，请利用 getKeyModifiersText()方法得到此字符串。

5

窗口与面版（Frame、Pane 与 Panel）的使用与介绍

Frame 与 Panel 是我们放置组件最常用到的容器，在本章中我们详细地介绍了这些容器的使用方法、使用时机等等。另外，我们也将介绍容器中组件的层次关系，使组件在排列上更容易控制，也可以制造出其他的特殊效果。另外我们也针对了 JInternalFrame、DesktopPane、SplitPane、JTabbedPane、JScrollPane、JScrollBar 等组件做了详细地介绍，当您读完此章节时，您将可以对版面的设计有一个很完整的概念。

深入浅出

5-1 JFrame 的使用

类层次结构图:

```
java.lang.Object
--java.awt.Component
--java.awt.Container
    --java.awt.Window
        --java.awt.Frame
            -- javax.swing.JFrame
```

JFrame 是在 Swing 中经常使用到的组件,您可以把它看成是最底层的容器,就如同是一个很大的水桶,水桶里可以装各种东西(例如 JLabel、JButton 等等),也可以装其他的小水桶(例如 Internal Frame、JPanel 等等)来摆放其他东西。而如何摆放当然就交给版面管理器啰!

在这章之前,我们已经在很多例子中使用 JFrame 了,不过在此章节,我们不但要更清楚知道 JFrame 的使用,还要了解整个 JFrame 的结构,这样读者就能够在简单的 JFrame 中作出更多样的变化。下面我们来看 JFrame 的构造函数,如表 5-1 所示。

表 5-1

JFrame 的构造函数

JFrame()

建立一个新的 JFrame,默认值是不可见的(Invisible)

JFrame(String title)

建立一个具有标题名称的 JFrame,默认值是不可见的(Invisible)

利用这些 JFrame 的构造函数,我们便可以很轻易地建立 JFrame 组件,如下范例:

范例 JFrame1.java (文件位于随书光盘目录 exam\ch5-JFrame1.java)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JFrame1 implements ActionListener
6  {
7      public JFrame1()
8      {
9          JFrame f = new JFrame("JFrameDemo");
10         Container contentPane = f.getContentPane();
11         JButton b = new JButton("Click me to get new Window");
12         b.addActionListener(this);
13         contentPane.add(b);
14         f.pack();
15         f.show();
16         f.addWindowListener(new WindowAdapter() {
17             public void windowClosing(WindowEvent e) {
18                 System.exit(0);
19             }
20         });
21     }
22 }
```

```
22
23     public void actionPerformed(ActionEvent e)
24     {
25         JFrame newf = new JFrame();
26         newf.setSize(200,200);
27         newf.show();
28     }
29
30     public static void main(String[] arg)
31     {
32         new JFrame1();
33     }
34 }
```

⊕ 说明:

- (1) 程序第 9 行, 我们建立一个具有标题的 JFrame 对象。
- (2) 程序第 10 行, 我们要在 JFrame 中加入其他组件必须先取得 Content Pane, 然后再加入组件到此 Content Pane 中。相对于 AWT, 若要在 AWT 中的 Frame 中加入某一个组件只要直接调用 add() 方法即可, 不需要先取得 Content Pane 再加入组件。Swing 这样的做法似乎多了一道手续, 却带来更强大、更有弹性的功能, 原因就在于 Swing 的 JFrame 具有层次 (Layer) 的概念, 可以让您在 JFrame 中放入的组件不会造成混乱。例如当一个 JFrame 有按钮 (JButton)、菜单 (JMenu)、快速菜单 (Popup Menu)、工具栏 (Toolbar) 与工具栏提示 (Tool Tips) 时, 到底哪个组件应该摆在什么组件的上面或下面, JFrame 都有办法处理, 下节我们将仔细说明这个问题。
- (3) 程序第 13 行, 将按钮 b 加入 Content Pane 中。
- (4) 程序第 15 行, 使 JFrame 变成可看见的 (Visible)。
- (5) 程序第 25 行, 产生一个没有标题的 JFrame。

程序运行结果如图 5-1 所示。



图 5-1

当按下按钮时会跳出一个没有标题的新窗口, 如图 5-2 所示。



图 5-2

5-2 Swing 的容器结构与 JLayeredPane 的使用

类层次结构图:

```
java.lang.Object
--java.awt.Component
--java.awt.Container
    --javax.swing.JComponent
        -- javax.swing.JLayeredPane
```

我们曾在第 3 章一开始时提到 Swing Top-Level 容器的层次结构与关系图, 相信读者对 Swing Top-Level 容器的结构已经有一个初步的概念。而在此节中, 我们将稍微复习一下以前的概念, 并实际应用 Swing 层次结构所提供的功能。

我们可以把 Swing 容器的结构看似如图 5-3 所示。

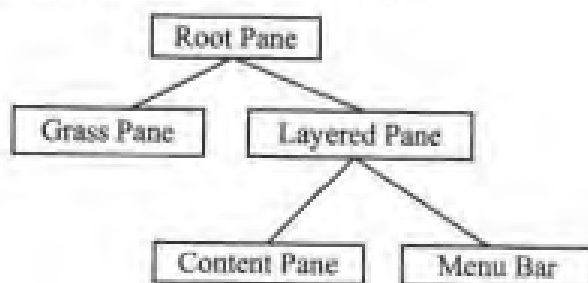


图 5-3

其中, Root Pane 可以看成是虚拟的容器, 包含着 Glass Pane、Layered Pane、Content Pane 与 Menu Bar。Swing 的容器包括 JApplet、JFrame、JDialog、JWindow 与 JInternalFrame 都是构造在此结构上, JApplet、JFrame、JDialog、JWindow 都是 heavyweight 容器, 只有 JInternalFrame 是 lightweight 容器。当我们要加入某个组件到 Swing 的容器中时, 并不是直接加入到 Root Pane, 而是加到 Root Pane 下面的某一成员(Layered Pane 或 Content Pane)。图 5-4 为 Root Pane 的实体结构图。

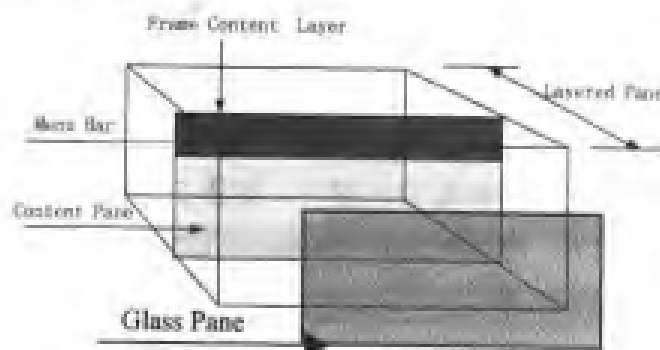


图 5-4

在上图中, 我们可以发现 Content Pane 与 Menu Bar 只是 Layered Pane 的其中一层, 我们称此层为 Frame-Content Layer。若你想知道 Frame-Content Layer 在 Layered Pane 的层次是什么, 您可以由 JLayeredPane 类中的 Frame_Content_Layer 类常数取得。

由此我们可以知道, Layered Pane 基本上可拥有非常多的“层”(Layer), 那么如何分辨

窗口与面版 (Frame、Pane 与 Panel) 的使用与介绍

哪一层是在哪一层的上面或下面呢? 答案是 Z-order。Z-order 本身具有两个整数值, 一个是代表层 (Layer) 的深度, 另一个代表同层的相关位置 (Position)。当 Z-order 的 Layer 数值越小时, 表示其位置就在越底层, 当 Z-order 的 Layer 数值越大时, 表示其位置就在越上层。在 JLayeredPane 类中, 共定义了 6 个 Z-order 的 Layer 常数供用户参考, 如下所示:

DEFAULT_LAYER: Z-order 的 Layer 数值为 0, 以整数对象 Integer(0) 来表示。一般我们加入的组件若没有标记是第几层, 默认值就是把组件放在此 Default Layer 中。

PALETTE_LAYER: Z-order 的 Layer 数值为 100, 以整数对象 Integer(100) 来表示。位于 Default Layer 之上, 一般用于放置可移动的工具栏 (Floatable Toolbar)。

MODAL_LAYER: Z-order 的 Layer 数值为 200, 以整数对象 Integer(200) 来表示。位于 Palette Layer 之上, 一般用于放置对话框 (Dialog Box)。

POPUP_LAYER: Z-order 的 Layer 数值为 300, 以整数对象 Integer(300) 来表示。位于 Modal Layer 之上, 一般用于快速菜单 (Popup Menu) 与工具栏提示 (Tool Tips) 中。

DRAG_LAYER: Z-order 的 Layer 数值为 400, 以整数对象 Integer(400) 来表示。位于 Popup Layer 之上, 一般用于拖曳组件使其在不同区域上。

FRAME_CONTENT_LAYER: Z-order 的 Layer 数值为 -30000, 以整数对象 Integer(-30000) 来表示。一般来说, Frame Content Layer 是最底层的 Layer, 用来表示 Content Pane 与 Menu Bar 的位置, 大部分的情况下我们不会更改到这个数值。

一般程序语言都会提供良好的 Z-order 自动管理机制, 当然 Java 也不例外。因此大部分的情况下我们都不会使用到 Z-order, 因为系统会自动帮我们管得好好的。用户只需要将所需的组件 (如按钮、菜单、工具栏提示等) 直接加入 Content Pane 即可, 根本不需要知道它们之间的顺序关系。但如果今天您必须处理对象之间的层次关系时, 例如在 Word 中您可以把某个绘图对象下推至下一层, 您就必须亲自处理 Z-order 的关系了。

下面我们来看如何利用 JLayeredPane 来控制对象的层次关系:

这个范例分别用 7 个 JLabel 对象层层相叠, 每个 JLabel 对象都有不同的 Z-order 数值, 形成 7 层相叠的效果。

范例 JLayeredPane1.java (文件位于随书光盘目录 exam\ch5\ JLayeredPane1.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JLayeredPanel extends JFrame
6  {
7      public JLayeredPanel()
8      {
9          super("JLayeredPane");
10         Integer[] layerConstants = { JLayeredPane.DEFAULT_LAYER,
11             JLayeredPane.PALETTE_LAYER, new Integer(101),
12             JLayeredPane.MODAL_LAYER, new Integer(201),
13             JLayeredPane.POPUP_LAYER, JLayeredPane.DRAG_LAYER };
14
15         Color[] colors = { Color.red, Color.blue,
16             Color.magenta, Color.cyan,
17             Color.yellow, Color.green,

```

```

18         Color.pink };
19
20     Point position = new Point(10,10);
21     JLabel[] label = new JLabel[7];
22     JLayeredPane layeredPane = getLayeredPane();
23
24     for (int i=0 ; i<7; i++)
25     {
26         label[i] = createLabel("第 "+(i+1)+" 层",colors[i],position);
27         position.x = position.x+20;
28         position.y = position.y+20;
29         layeredPane.add(label[i],layerConstants[i]);
30     }
31
32     setSize(new Dimension(280, 280));
33     show();
34     addWindowListener(new WindowAdapter() {
35         public void windowClosing(WindowEvent e) {
36             System.exit(0);
37         }
38     });
39 }
40
41 public JLabel createLabel(String content,
42                           Color color,
43                           Point position)
44 {
45     JLabel label = new JLabel(content,JLabel.CENTER);
46     label.setVerticalAlignment(JLabel.TOP);
47     label.setBackground(color);
48     label.setForeground(Color.black);
49     label.setOpaque(true);
50     label.setBounds(position.x, position.y, 100, 100);
51     return label;
52 }
53
54 public static void main(String[] arg)
55 {
56     new JLayeredPanel();
57 }
58 }

```

◆ 说明:

- (1) 程序第 10~13 行, 由小到大定义组件深度数值, 也就是定义 Z-order Layer 的大小。
- (2) 程序第 15~18 行, 定义每个 JLabel 的颜色。
- (3) 程序第 22 行, 取得窗口的 Layered Pane。
- (4) 程序第 29 行, 将组件 (JLabel) 放入 Layered Pane 中, 并给予深度 (Z-order Layer) 的数值。

程序运行结果如图 5-5 所示。

窗口与面版 (Frame、Pane 与 Panel) 的使用与介绍



图 5-5

您可以使用 `JLayeredPane` 类提供的 `getLayer()` 与 `setLayer()` 方法取得或设置组件的层次值。我们之前提到 `Content Pane` 一般视为 `Layered Pane` 的最底层, 因此若将组件加入 `Content Pane` 中, 此对象显示的层次是不是会比上面的 7 个 `JLabel` 对象还低呢? 我们来看下面这个例子:

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JLayeredPane2 extends JFrame
6  {
7      public JLayeredPane2()
8      {
9          super("JLayeredPane");
10         Integer[] layerConstants = {new Integer(-100),
11             JLayeredPane.PALETTE_LAYER, new Integer(101),
12             JLayeredPane.MODAL_LAYER, new Integer(201),
13             JLayeredPane.POPUP_LAYER, JLayeredPane.DRAG_LAYER };
14
15         Color[] colors = { Color.red, Color.blue,
16             Color.magenta, Color.cyan ,
17             Color.yellow, Color.green,
18             Color.pink };
19
20         Point position = new Point(10,10);
21         JButton[] button = new JButton[7];
22         JLayeredPane layeredPane = getLayeredPane();
23
24         for (int i=0 ; i<7; i++)
25         {
26             button[i] = createButton("第 "+(i+1)+" 层", colors[i], position);
27             position.x = position.x+20;
28             position.y = position.y+20;
29             layeredPane.add(button[i], layerConstants[i]);
30         }
31
32         Container contentPane = getContentPane();
33         contentPane.setLayout(new GridLayout(2,2));
34         JButton b1 = new JButton("按钮一");
35         contentPane.add(b1);
36         JButton b2 = new JButton("按钮二");
37         contentPane.add(b2);

```

```

38         JButton b3 = new JButton("按钮三");
39         contentPane.add(b3);
40         JButton b4 = new JButton("按钮四");
41         contentPane.add(b4);
42
43         System.out.println("Content Pane 层次值为: "+
44         layeredPane.getLayer(contentPane));
45         System.out.println("按钮一层次值为: "+
46         layeredPane.getLayer(b1));
47         System.out.println("按钮二层次值为: "+
48         layeredPane.getLayer(b2));
49         System.out.println("button[0]层次值为: "+
50         layeredPane.getLayer(button[0]));
51         System.out.println("button[1]层次值为: "+
52         layeredPane.getLayer(button[1]));
53         System.out.println("button[2]层次值为: "+
54         layeredPane.getLayer(button[2]));
55
56         setSize(new Dimension(280, 280));
57         show();
58         addWindowListener(new WindowAdapter() {
59             public void windowClosing(WindowEvent e) {
60                 System.exit(0);
61             }
62         });
63     }
64
65     public JButton createButton(String content,
66                                Color color,
67                                Point position)
68     {
69         JButton button = new JButton(content);
70         button.setVerticalAlignment(JButton.TOP);
71         button.setBackground(color);
72         button.setForeground(Color.black);
73         button.setOpaque(true);
74         button.setBounds(position.x, position.y, 100, 100);
75         return button;
76     }
77
78     public static void main(String[] arg)
79     {
80         new JLayeredPane2();
81     }
82 }

```

⊕ 说明:

(1) 此范例更改上一范例, 将 JLabel 换成 JButton, 并稍微更改 layerConstants 所设置的值。在程序中我们取得 JFrame 的 Content Pane, 并在此 Content Pane 中加入 4 个按钮。

(2) 程序第 44 行, 利用 getLayer() 方法取得组件的层次值。

程序运行结果如图 5-6 和图 5-7 所示。



图 5-6



图 5-7

从上面的例子可以看出，虽然“Button1”的层次数比“按钮一”低，但它却显示在“按钮一”之上，原因是“按钮一”是加在 Content Pane 中而不是加在 LayeredPane 上，因此显示时是以 Content Pane 与加在 Content Pane 上的组件来做层次的比较。

从上面的例子我们可以发现，Z-order Layer 数值越小的组件在越底层，也就会被 Z-order Layer 值较大的组件所覆盖。而读者也许会问，那如果两个组件都在同一层且相互重叠，怎么知道它们之间的层次关系呢？答案是利用 Z-order 的另外一个整数值：Position。Position 数值的关系跟 Z-order 的 Layer 数值恰好相反，在同一层中的对象，若 Position 数值越小者则在越上层，Position 数值越大者则在越下层。Position 的数值是从 -1~n-1，n 是指在同一层中组件的个数，数值 -1 代表最底层，意思跟 n-1 一样；数值 0 代表最上层。您可以使用 JLayeredPane 类提供的 moveToBack() 方法将组件推至 Position 为 -1 (最底端) 的位置，或是使用 moveToFront() 方法将组件推至 position 为 0 (最顶端) 的位置。我们来看下面这个范例：

范例 JLayeredPane2.java (文件位于随书光盘目录 exam\ch5\JLayeredPane2.java)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JLayeredPane2 extends JFrame
6  {
7      public JLayeredPane2()
8      {
9          super("JLayeredPane");
10
11          JLabel label1 = new JLabel("左 Label", JLabel.CENTER);
12          label1.setVerticalAlignment(JLabel.TOP);
13          label1.setBackground(Color.red);
14          label1.setForeground(Color.black);
15          label1.setOpaque(true);
16          label1.setBounds(20, 20, 150, 150);
17
18          JLabel label2 = new JLabel("右 Label", JLabel.CENTER);
19          label2.setVerticalAlignment(JLabel.TOP);
20          label2.setBackground(Color.green);
21          label2.setForeground(Color.black);
22          label2.setOpaque(true);
23          label2.setBounds(50, 50, 150, 150);
24
25          JLayeredPane layeredPane = getLayeredPane();
```



```

26         layeredPane.add(label1,new Integer(10),1);
27         layeredPane.add(label2,new Integer(10),0);
28
29         setSize(new Dimension(280, 280));
30         show();
31         addWindowListener(new WindowAdapter() {
32             public void windowClosing(WindowEvent e) {
33                 System.exit(0);
34             }
35         });
36     }
37
38     public static void main(String[] arg)
39     {
40         new JLayeredPane2();
41     }
42 }

```

⊕ 说明:

程序第 26、27 行中，在 Layered Pane 中加入 JLabel 对象，并指定为相同的 Layer，但不同的 Position。

⊕ 程序运行结果如下:

由于 label1 的 Position 大于 label2，且在同一层，因此根据我们上面所说的，label2 会在 label1 之上，如图 5-8 所示。

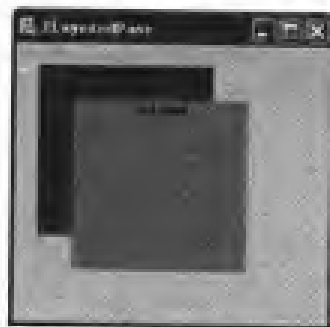


图 5-8

若我们把 label1 与 label2 的 Position 设置如下，则运行结果将是 label1 在 label2 之上:

```

layeredPane.add(label1,new Integer(10),0);
layeredPane.add(label2,new Integer(10),-1);

```

⊕ 程序运行结果如图 5-9 所示。

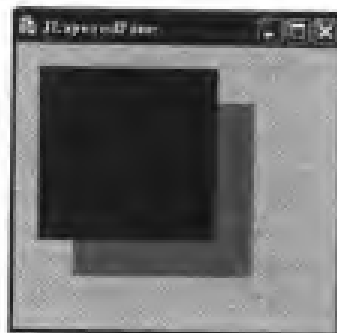


图 5-9

5-3 JInternalFrame 的使用

类层次结构图:

```
java.lang.Object
--java.awt.Component
--java.awt.Container
    --javax.swing.JComponent
        -- javax.swing.JInternalFrame
```

JInternalFrame 的使用跟 JFrame 几乎一样, 可以最大化、最小化、关闭窗口、加入菜单等功能; 唯一不同的是 JInternalFrame 是 lightweight component, 也就是说 JInternalFrame 不能单独出现, 必须依附在最上层组件上。由于这个特色, JInternalFrame 能够利用 Java 提供的 Look and Feel 功能作出完全不同于原有操作系统所提供的窗口外型, 也比 JFrame 更具有弹性。

一般我们会将 Internal Frame 加入 Desktop Pane 方便管理, Desktop Pane 是一种特殊的 Layered Pane, 用来建立虚拟桌面 (Virtual Desktop)。它可以显示并管理众多 Internal Frame 之间的层次关系。以下是 JDesktopPane 的类层次结构图:

类层次结构图:

```
java.lang.Object
--java.awt.Component
--java.awt.Container
    --javax.swing.JComponent
        --javax.swing.JLayeredPane
            -- javax.swing.JDesktop
```

我们先来看看如何构造 JInternalFrame 与 JDesktopPane, 再来看它跟 Desktop Pane 之间的关系。表 5-2 和表 5-3 是 JInternalFrame 与 JDesktopPane 的构造函数。

表 5-2

JInternalFrame 的构造函数	
JInternalFrame()	建立一个不能更改大小、不可关闭、不可最大最小化、也没有标题的 Internal Frame
JInternalFrame(String title)	建立一个不能更改大小、不可关闭、不可最大最小化, 但具有标题的 Internal Frame
JInternalFrame(String title, boolean resizable)	建立一个不可关闭、不可最大最小化, 但可变更大小且具有标题的 Internal Frame
JInternalFrame(String title, boolean resizable, boolean closable)	建立一个可关闭、可更改大小且具有标题, 但不可最大最小化的 Internal Frame
JInternalFrame(String title, boolean resizable, boolean closable, boolean maximizable)	建立一个可关闭、可更改大小、具有标题、可最大化, 但不可最小化的 Internal Frame
JInternalFrame(String title, boolean resizable, boolean closable, boolean maximizable, boolean iconifiable)	建立一个可关闭、可更改大小、具有标题、可最大化与最小化的 Internal Frame

JDesktopPane 的构造函数

JDesktopPane()

建立 JDesktopPane 组件

下面我们来看如何利用 JDesktopPane 来管理 JInternalFrame:

此范例一开始有一个按钮与一个 JDesktopPane, 当用户按下按钮时, 会在 Desktop Pane 中产生一个 JInternalFrame 并呈现 Active 状态; Internal Frame 中含有一个 JTextArea 与按钮对象。

范例 JInternalFrame1.java (文件位于随书光盘目录 exam\ch5\ JInternalFrame1.java)

```
1  import javax.swing.*;
2  import java.awt.event.*;
3  import java.awt.*;
4
5  public class JInternalFrame1 extends JFrame implements ActionListener{
6
7      JDesktopPane desktopPane;
8      int count = 1;
9
10     public JInternalFrame1() {
11         super("JInternalFrame1");
12         Container contentPane = this.getContentPane();
13         contentPane.setLayout(new BorderLayout());
14
15         JButton b = new JButton("Create New Internal Frames");
16         b.addActionListener(this);
17         contentPane.add(b, BorderLayout.SOUTH);
18
19         desktopPane = new JDesktopPane();
20         contentPane.add(desktopPane);
21
22         setSize(350, 350);
23         show();
24
25         addWindowListener(new WindowAdapter() {
26             public void windowClosing(WindowEvent e) {
27                 System.exit(0);
28             }
29         });
30     }
31
32     public void actionPerformed(ActionEvent e)
33     {
34         JInternalFrame internalFrame = new JInternalFrame(
35             "Internal Frame "+(count++), true, true, true, true);
36
37         internalFrame.setLocation( 20,20);
38         internalFrame.setSize(200,200);
39         internalFrame.setVisible(true);
40
41         Container icontentPane = internalFrame.getContentPane();
42         JTextArea textArea = new JTextArea();
```

窗口与面版 (Frame、Pane 与 Panel) 的使用与介绍

```

43         JButton b = new JButton("Internal Frame Button");
44         icontentPane.add(textArea,"Center");
45         icontentPane.add(b,"South");
46
47         desktopPane.add(internalFrame);
48
49         try {
50             internalFrame.setSelected(true);
51         } catch (java.beans.PropertyVetoException ex) {
52             System.out.println("Exception while selecting");
53         }
54     }
55
56     public static void main(String[] args) {
57         new JInternalFrame1();
58     }
59 }

```

⊕ 说明:

- (1) 程序第 19~20 行, 建立一个新的 JDesktopPane 并加入于 contentPane 中。
- (2) 程序第 16 行, 当用户按下按钮时, 将运行 actionPerformed() 中的程序代码。
- (3) 程序第 34~35 行, 产生一个可关闭、可改变大小、具有标题、可最大化与最小化的 Internal Frame。
- (4) 程序第 41 行, 取得 JInternalFrame 的 Content Pane, 用以加入新的组件。
- (5) 程序第 44~45 行, 将 JTextArea 与 JButton 对象加入 JInternalFrame 中。由此可知, JInternalFrame 加入组件的方式与 JFrame 是一模一样。
- (6) 程序第 47 行, 将 JInternalFrame 加入 JDesktopPane 中。如此一来, 即使产生很多 JInternalFrame, JDesktopPane 也能将它们之间的层次关系管理得相当良好。

⊕ 程序运行结果如图 5-10 所示。



图 5-10

5-4 JPanel 的使用

类层次结构图:

```
java.lang.Object
--java.awt.Component
--java.awt.Container
    --javax.swing.JComponent
        -- javax.swing.JPanel
```

Panel 也是 Java 中时常用到的容器之一, Panel 除了可以让组件加入外,有效的利用 Panel 也可以使版面管理更为容易。Swing 的 JPanel 支持 double buffering 的功能,使得 JPanel 在处理动画上更为流畅,较不会有画面闪烁的情况发生。表 5-4 为 JPanel 的构造函数。

表 5-4

JPanel 的构造函数	
JPanel()	建立一个具有 double buffering 功能的 JPanel, 默认的版面管理是 Flow Layout
JPanel(boolean isDoubleBuffered)	选择建立是否具有 double buffering 功能的 JPanel, 默认的版面管理是 Flow Layout
JPanel(LayoutManager layout)	建立一个具有 double buffering 功能的 JPanel, 可自定义版面管理器
JPanel(LayoutManager layout, boolean isDoubleBuffered)	选择建立是否具有 double buffering 功能的 JPanel, 并自定义版面管理器

在下面这个例子中我们用 JPanel 来排列 5 个 JLabel 对象, 您将可以发现利用 JPanel 搭配版面管理, 可以做出更复杂的排列效果:

范例 JPanel1.java (文件位于随书光盘目录 exam\ch5\ JPanel1.java)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JPanel1
6  {
7      public JPanel1()
8      {
9          JFrame f = new JFrame("JPanelDemo");
10         Container contentPane = f.getContentPane();
11         contentPane.setLayout(new GridLayout(2,1));
12         JLabel[] label = new JLabel[5];
13
14         for(int i=0; i<label.length ; i++)
15         {
16             label[i] = new JLabel("Label "+(i+1),JLabel.CENTER);
17             label[i].setBackground(Color.lightGray);
18             label[i].setBorder(BorderFactory.createEtchedBorder());
19             label[i].setOpaque(true);
20
21         }
22 }
```

窗口与面版 (Frame、Pane 与 Panel) 的使用与介绍

```

23     JPanel panel1 = new JPanel(new GridLayout(1,1));
24     panel1.add(label[0]);
25
26     JPanel panel2 = new JPanel(new GridLayout(1,2));
27
28     JPanel panel3 = new JPanel(new GridLayout(1,2));
29     panel3.add(label[1]);
30     panel3.add(label[2]);
31
32     JPanel panel4 = new JPanel(new GridLayout(2,1));
33     panel4.add(label[3]);
34     panel4.add(label[4]);
35
36     panel2.add(panel3);
37     panel2.add(panel4);
38
39     contentPane.add(panel1);
40     contentPane.add(panel2);
41
42     f.pack();
43     f.show();
44     f.addWindowListener(new WindowAdapter() {
45         public void windowClosing(WindowEvent e) {
46             System.exit(0);
47         }
48     });
49 }
50
51 public static void main(String[] arg)
52 {
53     new JPanell();
54 }
55 }

```

⊕ 说明:

(1) 整个组件摆设结构如图 5-11 所示。

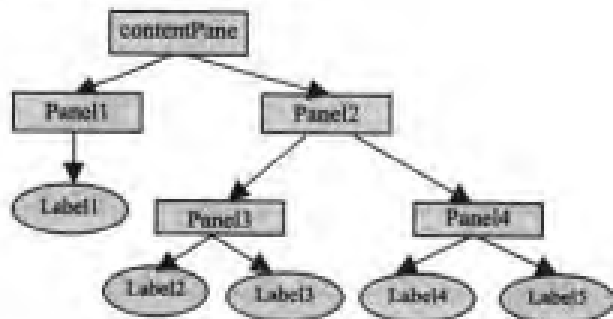


图 5-11

- (1) 程序第 14~21 行中, 我们利用循环产生 5 个 JLabel 对象, 并设置背景颜色与边框, 此部分在下一章将有详细的介绍。
- (2) 程序第 19 行, setOpaque (ture) 方法的目的是让组件变成不透明状, 这样我们在 JLabel 上所设置的颜色才能显示出来。

由程序运行结果如图 5-12 所示。



图 5-12

由上图可知，利用 JPanel 可以使版面的排列方式更生动。若没有 JPanel 的帮助，想直接由 contentPane 排列成如上图所示，就必须借助复杂的 GridBagLayout 版面管理器了。

5-5 JSplitPane 的使用

类层次结构图：

```
java.lang.Object
--java.awt.Component
--java.awt.Container
    --javax.swing.JComponent
        -- javax.swing.JSplitPane
```

Split Pane（分割面版）一次可将两个组件同时显示在两个显示区中，若您想要同时在多个显示区显示组件，您便必须同时使用多个 Split Pane。JSplitPane 提供两个常数让您设置到底是要水平分割还是垂直分割，这两个常数分别是：HORIZONTAL_SPLIT 与 VERTICAL_SPLIT。除了这两个重要的常数外，JSplitPane 还提供了许多类常数让您使用，我们会通过下面的例子介绍比较常用的类常数，其余信息读者可自行参考一般的 Java API 文件。

我们先来看看 JSplitPane 的构造函数，如表 5-5 所示。

表 5-5

JSplitPane 的构造函数	
JSplitPane()	建立一个新的 JSplitPane，里面含有两个默认按钮，并以水平方向排列，但没有 Continuous Layout 功能
JSplitPane(int newOrientation)	建立一个指定水平或垂直方向切割的 JSplitPane，但没有 Continuous Layout 功能
JSplitPane(int newOrientation, boolean newContinuousLayout)	建立一个指定水平或垂直方向切割的 JSplitPane，且指定是否具有 Continuous Layout 功能
JSplitPane(int newOrientation, boolean newContinuousLayout, Component newLeftComponent, Component newRightComponent)	建立一个指定水平或垂直方向切割的 JSplitPane，且指定显示区所要显示的组件，并设置是否具有 Continuous Layout 功能
JSplitPane(int newOrientation, Component newLeftComponent, Component newRightComponent)	建立一个指定水平或垂直方向切割的 JSplitPane，且指定显示区所要显示的组件，但没有 Continuous Layout 功能

窗口与面版 (Frame、Pane 与 Panel) 的使用与介绍

上面所说的 Continuous Layout 功能是指当您拖曳切割面版的分隔线时, 窗口内的组件是否会随着分隔线的拖曳而动态地改变大小。newContinuousLayout 是一个 boolean 值, 若设为 true, 则组件大小会随着分隔线的拖曳而一起改动; 若设为 false, 则组件大小在分隔线停止改动时才确定。你也可以使用 JSplitPane 中的 setContinuousLayout() 方法来设置此项目。

下面我们来看 JSplitPane 的例子:

范例 JSplitPane1.java (文件位于随书光盘目录 exam\ch5\JSplitPane1.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JSplitPanel
6  {
7      public JSplitPanel()
8      {
9          JFrame f = new JFrame("JSplitPaneDemo");
10         Container contentPane = f.getContentPane();
11         JLabel label1 = new JLabel("Label 1",JLabel.CENTER);
12         label1.setBackground(Color.green);
13         label1.setOpaque(true);
14         JLabel label2 = new JLabel("Label 2",JLabel.CENTER);
15         label2.setBackground(Color.pink);
16         label2.setOpaque(true);
17         JLabel label3 = new JLabel("Label 3",JLabel.CENTER);
18         label3.setBackground(Color.yellow);
19         label3.setOpaque(true);
20
21         JSplitPane splitPanel = new JSplitPane(JSplitPane.HORIZONTAL_
22         SPLIT,false,label1,label2);
23         splitPanel.setDividerLocation(0.3);
24         splitPanel.setOneTouchExpandable(true);
25         splitPanel.setDividerSize(10);
26
27         JSplitPane splitPane2 = new JSplitPane(JSplitPane.VERTICAL_
28         SPLIT,true,splitPanel,label3);
29         splitPane2.setDividerLocation(35);
30         splitPane2.setOneTouchExpandable(false);
31         splitPane2.setDividerSize(5);
32
33         contentPane.add(splitPane2);
34
35         f.setSize(250,200);
36         f.show();
37         f.addWindowListener(new WindowAdapter() {
38             public void windowClosing(WindowEvent e) {
39                 System.exit(0);
40             }
41         });
42     }
43
44     public static void main(String[] arg)
45     {
46         new JSplitPanel();

```



```

47     }
48 }

```

说明：

- (1) 程序第 21~22 行，我们加入 label1 与 label2 至 splitPanel1 中，并设置此 splitPanel1 为水平分割且具有 Continuous Layout 的功能。
- (2) 程序第 23 行，设置 splitPanel1 的分隔线位置，0.3 是相对于 splitPanel1 的大小而定，因此这个值的设置范围在 0.0~1.0 中。若您使用整数值来设置 splitPane 的分隔线位置，如程序第 29 行所示，则所定义的值以 pixel 为计算单位。
- (3) 程序第 24 行，设置 JSplitPane 是否可以展开或收起（如同文件总管一般），设为 true 表示打开此功能。
- (4) 程序第 25 行，设置分隔线宽度的大小，以 pixel 为计算单位。
- (5) 程序第 27~28 行，我们加入 splitPanel1 与 label3 至 splitPane2 中，并设置此 splitPane2 为垂直分割但不具有 Continuous Layout 的功能。

程序运行结果如图 5-13 所示。

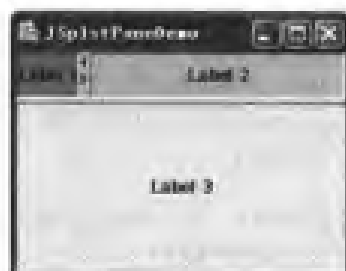


图 5-13

您可以发现，在 Label1 与 Label2 中的分隔线具有展开或收起的箭头，且此分隔线也比 Label2 与 Label3 间的分隔线粗。当您拖动 Label1 与 Label2 中的分隔线时，Label1 与 Label2 的大小不会随着变动，直到分隔线位置确定后才会确定这两个组件的大小。Label2 与 Label3 间的分隔线恰好相反，因为我们设置 splitPane2 的 Continuous Layout 为 true。

5-6 JTabbedPane 的使用

类层次结构图：

```

java.lang.Object
--java.awt.Component
--java.awt.Container
    --javax.swing.JComponent
        -- javax.swing.JTabbedPane

```

JTabbedPane 就如同放置文件的文件夹一般，当用户想要看哪一份文件时，只要循着标签上的说明文字来找寻，找到之后将它显示出来即可。利用这个功能，用户可以有效的管理自己的信息或文件，也可以使版面看起来干净许多。表 5-6 是 JTabbedPane 的构造函数。

窗口与面版 (Frame、Pane 与 Panel) 的使用与介绍

表 5-6

JTabbedPane 的构造函数

JTabbedPane()

建立一个空的 JTabbedPane 对象

JTabbedPane(int tabPlacement)

建立一个空的 JTabbedPane 对象, 并指定摆放位置, 如 TOP、BOTTOM、LEFT 或 RIGHT

以下我们来看个 JTabbedPane 的例子。下面这个范例中我们会在 Content Pane 中加入一个 JTabbedPane, 此 JTabbedPane 含有 3 个标签, 可以分别显示出 1 个 ImageIcon 图形与 2 个 JLabel 对象:

范例 JPanel1.java (文件位于随书光盘目录 exam\ch5\JPanel1.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JTabbedPane1
6  {
7      public JTabbedPane()
8      {
9          JFrame f = new JFrame("JTabbedPaneDemo");
10         Container contentPane = f.getContentPane();
11
12         JLabel label1 = new JLabel(new ImageIcon(".\\icons\\flower.jpg"));
13         JPanel panel1 = new JPanel();
14         panel1.add(label1);
15
16         JLabel label2 = new JLabel("Label 2",JLabel.CENTER);
17         label2.setBackground(Color.pink);
18         label2.setOpaque(true);
19         JPanel panel2 = new JPanel();
20         panel2.add(label2);
21
22         JLabel label3 = new JLabel("Label 3",JLabel.CENTER);
23         label3.setBackground(Color.yellow);
24         label3.setOpaque(true);
25         JPanel panel3 = new JPanel();
26         panel3.add(label3);
27
28         JTabbedPane tabbedPane = new JTabbedPane(JTabbedPane.BOTTOM);
29         tabbedPane.addTab("Picture",new
30         ImageIcon(".\\icons\\eye.jpg"),panel1,"图案");
31         tabbedPane.addTab("Label 2",panel2);
32         tabbedPane.addTab("Label 3",null,panel3,"label");
33
34         contentPane.add(tabbedPane);
35
36         f.pack();
37         f.show();
38         f.addWindowListener(new WindowAdapter() {
39             public void windowClosing(WindowEvent e) {
40                 System.exit(0);

```

```
41         }  
42     }  
43 }  
44  
45 public static void main(String[] arg)  
46 {  
47     new JTabbedPane();  
48 }  
49 }  
50
```

⊕ 说明:

- (1) 程序第 28 行, 建立一个 JTabbedPane 对象, 并使其摆放在窗口底端。
- (2) 程序第 29~30 行, 在 tabbedPane 上加入一个标签, 标签名称是 “Picture”, 标签上有一个 Icon 的图案, 标签里的内容是 panel1, 标签提示文字为 “图案”。
- (3) 程序第 31 行, 在 tabbedPane 上加入一个标签, 标签名称是 “Label 2”, 标签里的内容是 panel2, 没有标签图案与标签提示文字。
- (4) 程序第 32 行, 在 tabbedPane 上加入一个标签, 标签名称是 “Label 3”, 标签里的内容是 panel3, 标签提示文字为 “label”, 但没有标签图案。

程序运行结果如图 5-14 所示。



图 5-14

若按下 Label 3 标签, 则内容变成如图 5-15 所示。



图 5-15

窗口与面版 (Frame、Pane 与 Panel) 的使用与介绍

以上是简单的 JTabbedPane 的范例介绍, 接着我们来说明 JTabbedPane 的事件处理模式。

JTabbedPane 以处理 ChangeEvent 事件为主。每当在 JTabbedPane 中选换标签时, 都会产生 ChangeEvent 事件, 因此若要处理选换标签所对应的操作, 就必须实现 ChangeListener 这个 Interface。另外, JTabbedPane 上的每个标签都有索引值 (Index), 一般若没有加以设置, 索引值从左到右依次是 0, 1, 2……依此类推, 因此上例中, Picture 的索引值为 0, Label 2 的索引值为 1。

下面这个例子中, 我们新增一个按钮, 当用户按下此按钮时可以在 JTabbedPane 上增加一个标签, 且我们限制如果要看右边标签的内容, 必须先看过左边标签的内容, 否则标签将显示 Disable 状态。

范例 JTabbedPane2.java (文件位于随书光盘目录 exam\ch5\JTabbedPane2.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.event.*;
4  import javax.swing.*;
5
6  public class JTabbedPane2 implements ActionListener,ChangeListener
7  {
8      int index = 0; // 索引值初始值设为 0, 代表第一个标签
9      int newNumber = 1;
10     JTabbedPane tabbedPane = null;
11
12     public JTabbedPane2()
13     {
14         JFrame f = new JFrame("JTabbedPaneDemo");
15         Container contentPane = f.getContentPane();
16
17         JLabel label1 = new JLabel(new ImageIcon(".\\icons\\flower.jpg"));
18         JPanel panel1 = new JPanel();
19         panel1.add(label1);
20
21         JLabel label2 = new JLabel("Label 2",JLabel.CENTER);
22         label2.setBackground(Color.pink);
23         label2.setOpaque(true);
24         JPanel panel2 = new JPanel();
25         panel2.add(label2);
26
27         JLabel label3 = new JLabel("Label 3",JLabel.CENTER);
28         label3.setBackground(Color.yellow);
29         label3.setOpaque(true);
30         JPanel panel3 = new JPanel();
31         panel3.add(label3);
32
33         tabbedPane = new JTabbedPane();
34         tabbedPane.setTabPlacement(JTabbedPane.BOTTOM);
35         tabbedPane.addChangeListener(this);
36         tabbedPane.addTab("Picuure",new
37         ImageIcon(".\\icons\\eye.jpg"),panel1,"图案");
38         tabbedPane.addTab("Label 2",panel2);
39         tabbedPane.addTab("Label 3",null,panel3,"label");
40         tabbedPane.setEnabledAt(2,false); //设 Label 3 标签为 Disable 状态

```

```

41
42     JButton b = new JButton("新增标签");
43     b.addActionListener(this);
44     contentPane.add(b, BorderLayout.NORTH);
45     contentPane.add(tabbedPane, BorderLayout.CENTER);
46
47     f.pack();
48     f.show();
49     f.addWindowListener(new WindowAdapter() {
50         public void windowClosing(WindowEvent e) {
51             System.exit(0);
52         }
53     });
54 }
55
56 public void stateChanged(ChangeEvent e)
57 {
58     if (index != tabbedPane.getSelectedIndex())
59     {
60         if (index < tabbedPane.getTabCount()-1)
61             tabbedPane.setEnabledAt(index+1, true);
62     }
63     index = tabbedPane.getSelectedIndex();
64 }
65
66 public void actionPerformed(ActionEvent e1)
67 {
68     JPanel panel = new JPanel();
69     tabbedPane.addTab("new "+newNumber, panel);
70     tabbedPane.setEnabledAt(newNumber+2, false);
71     newNumber++;
72     tabbedPane.validate();
73 }
74
75 public static void main(String[] arg)
76 {
77     new JTabbedPane2();
78 }
79 }

```

⊕ 说明:

- (1) 程序第 3 行, 由于 `ChangeEvent` 是属于 `Swing` 的事件, 而不是 `AWT` 的事件, 因此 `import Swing` 的事件类来处理 `ChangeEvent` 事件。
- (2) 程序第 34 行, 设置标签摆放的位置。
- (3) 程序第 40 行, 一开始我们设置标签 `Label1` 与标签 `Label2` 为 `Enable` 状态, 标签 `Label3` 之后 (包括新增的标签) 都为 `Disable` 状态。
- (4) 程序第 56~64 行, 实现 `ChangeListener` 界面, 目的是使若左边的标签有点选过, 右边的标签才会显示 `Enable` 状态。`getSelectedIndex()` 方法可返回目前点选标签的 `index` 值, `getTabCount()` 方法可返回 `JTabbedPane` 上目前共有几个标签, 而 `setEnabledAt()` 方法则是使某个标签的状态为 `Enable` 或为 `Disable` (`true` 为 `Enable`, `false` 为 `Disable`)。

窗口与面版 (Frame、Pane 与 Panel) 的使用与介绍

(5) 程序第 66~73 行, 实现 ActionListener 界面, 当用户按下“新增标签”按钮时, 就会在 tabbedPane 上新增一个为 Disable 状态的标签。

⊕ 程序运行结果如下:

“Label1”与“Label2”的初始状态为 Enable, “Label3”为 Disable 状态, 如图 5-16 所示。



图 5-16

按两次按钮新增两个标签, 如图 5-17 所示。



图 5-17

若依次点选过“Label2”与“Label3”后, 如图 5-18 所示。



图 5-18

5-7 JScrollPane 的使用

类层次结构图:

```
java.lang.Object
--java.awt.Component
--java.awt.Container
    --javax.swing.JComponent
        -- javax.swing.JScrollPane
```

ScrollPane (滚动面版) 可以用来滚动窗口。当您使用浏览器浏览窗口时, 应该经常会发现窗口内的内容大于窗口的大小, 因此浏览器窗口的右边或下边就会出现一个滚动轴, 供用户拖曳来看到完整的内容。JScrollPane 就具有这样的功能, 且非常容易使用。事实上, JScrollPane 是由 JViewport 与 JScrollBar 所组成。JViewport 主要负责显示内容区域的大小, 形状为一个平面矩形; 而 JScrollBar 则产生窗口滚动轴, 让用户可以看到更多的内容。当用户在使用 JScrollPane 时, 通常不会直接使用到 JViewport 与 JScrollBar, 系统会依照用户的设置来控制这些组件, 可说是相当方便。

表 5-7 是 JScrollPane 的构造函数。

表 5-7

JScrollPane 的构造函数	
JScrollPane()	建立一个空的 JScrollPane 对象
JScrollPane(Component view)	建立一个新的 JScrollPane 对象, 当组件内容大于显示区域时会自动产生滚动轴
JScrollPane(Component view, int vsbPolicy, int hsbPolicy)	建立一个新的 JScrollPane 对象, 里面含有显示组件, 并设置滚动轴出现时机
JScrollPane(int vsbPolicy, int hsbPolicy)	建立一个新的 JScrollPane 对象, 里面不含显示组件, 但设置滚动轴出现时机

JScrollPane 可利用下面这些参数来设置滚动轴的出现时机, 这些参数是定义在 ScrollPaneConstants Interface 中, 而 JScrollPane 类实现此界面, 因此也就能使用这些参数:

HORIZONTAL_SCROLLBAR_ALWAYS: 显示水平滚动轴。

HORIZONTAL_SCROLLBAR_AS_NEEDED: 当组件内容的水平区域大于显示区域时出现水平滚动轴。

HORIZONTAL_SCROLLBAR_NEVER: 不显示水平滚动轴。

VERTICAL_SCROLLBAR_ALWAYS: 显示垂直滚动轴。

VERTICAL_SCROLLBAR_AS_NEEDED: 当组件内容的垂直区域大于显示区域时出现垂直滚动轴。

VERTICAL_SCROLLBAR_NEVER: 不显示垂直滚动轴。

我们先看一个简单的 JScrollPane 例子。下面例子中左边会有三个按钮, 右边是一个 JScrollPane, 用户可以选择是否出现滚动轴:

范例 JScrollPane1.java (文件位于随书光盘目录 exam\ch5\ JScrollPane1.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JScrollPane1 implements ActionListener
6  {
7      JScrollPane scrollPane;
8
9      public JScrollPane1()
10     {
11         JFrame f = new JFrame("JScrollPaneDemo");
12         Container contentPane = f.getContentPane();
13
14         JLabel label1 = new JLabel(new ImageIcon(".\\icons\\flower.jpg"));
15         JPanel panel1 = new JPanel();
16         panel1.add(label1);
17         scrollPane = new JScrollPane(panel1);
18
19         JPanel panel2 = new JPanel(new GridLayout(3,1));
20         JButton b = new JButton("显示水平滚动轴");
21         b.addActionListener(this);
22         panel2.add(b);
23         b = new JButton("不显示水平滚动轴");
24         b.addActionListener(this);
25         panel2.add(b);
26         b = new JButton("适时显示水平滚动轴");
27         b.addActionListener(this);
28         panel2.add(b);
29
30         contentPane.add(panel2, BorderLayout.WEST);
31         contentPane.add(scrollPane, BorderLayout.CENTER);
32
33         f.setSize(new Dimension(350,220));
34         f.show();
35         f.addWindowListener(new WindowAdapter() {
36             public void windowClosing(WindowEvent e) {
37                 System.exit(0);
38             }
39         });
40     }
41
42     public void actionPerformed(ActionEvent e)
43     {
44         if (e.getActionCommand().equals("显示水平滚动轴"))
45             scrollPane.setHorizontalScrollBarPolicy(
46                 JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
47         if (e.getActionCommand().equals("不显示水平滚动轴"))
48             scrollPane.setHorizontalScrollBarPolicy(
49                 JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
50         if (e.getActionCommand().equals("适时显示水平滚动轴"))
51             scrollPane.setHorizontalScrollBarPolicy(
52                 JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
53         scrollPane.revalidate();

```



```
54     }  
55     public static void main(String[] arg)  
56     {  
57         new JScrollPane();  
58     }  
59 }
```

⊕ 说明:

- (1) 程序第 17 行, 新增一个 JScrollPane 对象, 并将 panel1 放入 JScrollPane 中, 若 panel1 组件的大小大于窗口的大小则会自动显示出 ScrollBar。
- (2) 程序第 42~54 行, 实现 ActionListener 界面, 当用户按下“显示水平滚动轴”按钮时, 则设置水平滚动轴参数为 HORIZONTAL_SCROLLBAR_ALWAYS, 若用户按下“不显示水平滚动轴”按钮时, 则设置水平滚动轴参数为 HORIZONTAL_SCROLLBAR_NEVER, 若用户按下“适时显示水平滚动轴”按钮时, 则设置水平滚动轴参数为 HORIZONTAL_SCROLLBAR_AS_NEEDED。
- (3) 程序第 53 行, 重新显示 JScrollPane 形状。

程序运行结果如下:

⊕ 初始状态, 如图 5-19 所示。

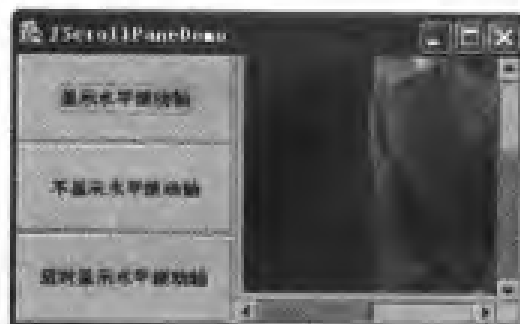


图 5-19

因为图片大于窗口所设置的大小, 因此垂直与水平滚动轴均会出现, 若按下“不显示水平滚动轴”按钮时, 则水平滚动轴会消失, 如图 5-20 所示。



图 5-20

接下来我们再看看 JScrollPane 还有哪些常用的功能。JScrollPane 除了可让您滚动滚动轴外, 它还可以设置表头 (Header) 名称、边角 (Corner) 图案与 ScrollPane 外框。我们修改上面的例子, 使 JScrollPane 更具有变化:

范例 JScrollPane2.java (文件位于随书光盘目录 exam\ch5\ JScrollPane2.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import javax.swing.border.*;
5
6  public class JScrollPane2 implements ActionListener
7  {
8      JScrollPane scrollPane;
9
10     public JScrollPane2()
11     {
12         JFrame f = new JFrame("JScrollPaneDemo");
13         Container contentPane = f.getContentPane();
14
15         JLabel label1 = new JLabel(new ImageIcon(".\\icons\\flower.jpg"));
16         JPanel panel1 = new JPanel();
17         panel1.add(label1);
18         scrollPane = new JScrollPane();
19         scrollPane.setViewportView(panel1);
20         scrollPane.setColumnHeaderView(new JLabel("水平表头"));
21         scrollPane.setRowHeaderView(new JLabel("垂直表头"));
22         scrollPane.setViewportBorder(
23             BorderFactory.createBevelBorder(BevelBorder.LOWERED));
24         scrollPane.setCorner(JScrollPane.UPPER_LEFT_CORNER,
25             new JLabel(new ImageIcon(".\\icons\\glass.jpg")));
26         scrollPane.setCorner(JScrollPane.UPPER_RIGHT_CORNER,
27             new JLabel(new ImageIcon(".\\icons\\glass.jpg")));
28
29         JPanel panel2 = new JPanel(new GridLayout(3,1));
30         JButton b = new JButton("显示水平滚动轴");
31         b.addActionListener(this);
32         panel2.add(b);
33         b = new JButton("不显示水平滚动轴");
34         b.addActionListener(this);
35         panel2.add(b);
36         b = new JButton("适时显示水平滚动轴");
37         b.addActionListener(this);
38         panel2.add(b);
39
40         contentPane.add(panel2, BorderLayout.WEST);
41         contentPane.add(scrollPane, BorderLayout.CENTER);
42
43         f.setSize(new Dimension(350,220));
44         f.show();
45         f.addWindowListener(new WindowAdapter() {
46             public void windowClosing(WindowEvent e) {
47                 System.exit(0);
48             }
49         });
50     }
51
52     public void actionPerformed(ActionEvent e)
53     {

```

```

54         if (e.getActionCommand().equals("显示水平滚动轴"))
55             scrollPane.setHorizontalScrollBarPolicy(
56                 JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
57         if (e.getActionCommand().equals("不显示水平滚动轴"))
58             scrollPane.setHorizontalScrollBarPolicy(
59                 JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
60         if (e.getActionCommand().equals("适时显示水平滚动轴"))
61             scrollPane.setHorizontalScrollBarPolicy(
62                 JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
63         scrollPane.revalidate();
64     }
65     public static void main(String[] arg)
66     {
67         new JScrollPane2();
68     }
69 }

```

◆ 说明：

- (1) 程序第 19 行，设置窗口显示组件为 `panell`。
- (2) 程序第 20~21 行，设置 `scrollPane` 的水平与垂直表头。
- (3) 程序第 22~23 行，设置 `scrollPane` 的边框为凹陷立体边框。边框（Border）部分我们将在第 6 章有详细介绍。
- (4) 程序第 24~27 行，设置 `scrollPane` 的边角图案，由于 `JScrollPane` 为矩形形状，因此就有 4 个位置来摆放边角（Corner）组件，这 4 个地方分别是左上、左下、右上、右下，对应的参数分别如下：

- `JScrollPane.UPPER_LEFT_CORNER`
- `JScrollPane.LOWER_LEFT_CORNER`
- `JScrollPane.UPPER_RIGHT_CORNER`
- `JScrollPane.LOWER_RIGHT_CORNER`

整个 `JScrollPane` 结构与程序运行结果如图 5-21 所示。

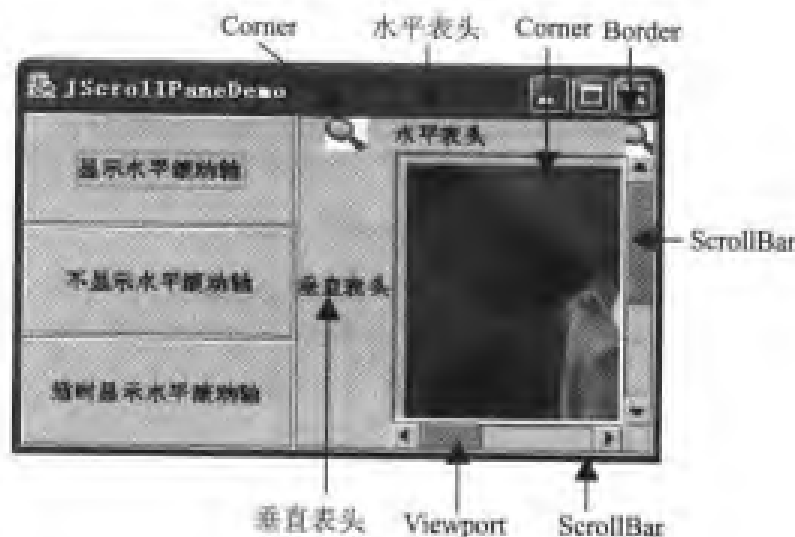


图 5-21

5-8 JScrollBar 的使用

类层次结构图:

```
java.lang.Object
--java.awt.Component
--java.awt.Container
    --javax.swing.JComponent
        -- javax.swing.JScrollBar
```

在上一节中我们看到 JScrollPane 利用 ScrollBar 的功能使它可以利用滚动轴滚动窗口，乍看之下我们并不会直接使用到 JScrollBar 的方法，因为 JScrollPane 都帮我们处理得很好。但如果我们想对滚动轴做更细的设置，例如在拖曳时一次滚动多少区域等，就必须了解 JScrollBar 所提供的功能了。JScrollBar 在处理窗口的滚动时并不像 JScrollPane 那么容易，看起来也比 JScrollPane 简单许多，因此通常在使用中我们会取一些 JScrollBar 所提供的功能，来补足我们对 JScrollPane 的需要，而不会直接拿 JScrollBar 来做滚动轴的操作。表 5-8 是 JScrollBar 的构造函数。

表 5-8

JScrollBar 的构造函数

JScrollBar()	建立一个垂直的滚动轴，默认参数值分别是: minimum = 0, maximum = 100, value = 0, extent = 10
JScrollBar(int orientation)	建立一个指定方向的滚动轴，默认参数值分别是: minimum = 0, maximum = 100, value = 0, extent = 10
JScrollBar(int orientation, int value, int extent, int min, int max)	建立一个指定方向的滚动轴，并设置 value、extent、minimum 与 maximum 的参数值

JScrollBar 4 个参数的意义如下:

value: JScrollBar 一开始的起始位置，若设为 0 表示在滚动轴的最顶端。

extent: 延伸区，限制滚动轴可滚动的范围。例如，若 minimum 值设为 0，maximum 值设为 100，而 extent 值设为 20，则滚动轴可滚动的区域大小为 $100 - 20 - 0 = 80$ 个刻度，滚动的范围从 0~80。若 minimum 值设为 20，maximum 值设为 100，而 extent 值设为 30，则滚动轴可滚动的区域大小为 $100 - 30 - 20 = 50$ 个刻度，滚动的范围从 20~70。因此可知，延伸区设得越大，可滚动的范围就越小。

minimum: 设置最小刻度值。

maximum: 设置最大刻度值。

JScrollBar 最常用到的就是 AdjustmentEvent 事件，当用户拖曳滚动轴时就会触发此事件。因此若要处理这类事件，就必须实现 AdjustmentListener 界面。此界面定义了一个 adjustmentValueChanged() 方法，实现此方法就能够得到滚动轴的相关信息。

举一个 JScrollBar 的简单例子。下面的范例中我们在 Content Pane 中分别加入水平与垂直滚动轴，并设置相关的参数值。当用户拖曳滚动轴时，Content Pane 上的 JLabel 会显示滚动的信息:

```
1 import java.awt.*;
2 import java.awt.event.*;
```

```
3    import javax.swing.*;
4
5    public class JScrollBar1 implements AdjustmentListener
6    {
7        JScrollBar scrollBar1;
8        JScrollBar scrollBar2;
9        JPanel panel1;
10       JLabel label2 = new JLabel("刻度: ", JLabel.CENTER);
11
12       public JScrollBar1()
13       {
14           JFrame f = new JFrame("JScrollBarDemo");
15           Container contentPane = f.getContentPane();
16
17           JLabel label1 = new JLabel(new ImageIcon(".\\icons\\flower.jpg"));
18           panel1 = new JPanel();
19           panel1.add(label1);
20           scrollBar1 = new JScrollBar(JScrollBar.VERTICAL, 10, 10, 0, 100);
21           scrollBar1.setUnitIncrement(1);
22           scrollBar1.setBlockIncrement(10);
23           scrollBar1.addAdjustmentListener(this);
24
25           scrollBar2 = new JScrollBar();
26           scrollBar2.setOrientation(JScrollBar.HORIZONTAL);
27           scrollBar2.setValue(0);
28           scrollBar2.setVisibleAmount(20);
29           scrollBar2.setMinimum(10);
30           scrollBar2.setMaximum(60);
31           scrollBar2.setBlockIncrement(5);
32           scrollBar2.addAdjustmentListener(this);
33
34           contentPane.add(panel1, BorderLayout.CENTER);
35           contentPane.add(scrollBar1, BorderLayout.EAST);
36           contentPane.add(scrollBar2, BorderLayout.SOUTH);
37           contentPane.add(label2, BorderLayout.NORTH);
38
39           f.setSize(new Dimension(200, 200));
40           f.show();
41           f.addWindowListener(new WindowAdapter() {
42               public void windowClosing(WindowEvent e) {
43                   System.exit(0);
44               }
45           });
46       }
47
48       public void adjustmentValueChanged(AdjustmentEvent e)
49       {
50           if ((JScrollBar)e.getSource() == scrollBar1)
51               label2.setText("垂直刻度: "+e.getValue());
52           if ((JScrollBar)e.getSource() == scrollBar2)
53               label2.setText("水平刻度: "+e.getValue());
54       }
55       public static void main(String[] arg)
56       {
57           new JScrollBar1();
```

```
58     }
59 }
```

⊕ 说明:

- (1) 程序第 20 行, 产生一个垂直滚动轴, 默认滚动轴位置在 10 刻度的地方, extent 值设为 10, minimum 值为 0, maximum 值为 100, 因此滚动轴一开始在刻度 10 的位置上, 可滚动的区域大小为 $100-10-0=90$ 个刻度, 滚动范围在 0~90 中。
- (2) 程序第 21 行, 设置拖曳滚动轴时, 滚动轴刻度一次的变化量。
- (3) 程序第 22 行, 设置当鼠标在滚动轴列上按一下时, 滚动轴一次所跳的区块大小。
- (4) 程序第 25 行, 建立一个空的 JScrollBar。
- (5) 程序第 26 行, 设置滚动轴方向为水平方向。
- (6) 程序第 27~31 行, 设置默认滚动轴位置在 0 刻度的地方, extent 值设为 20, minimum 值为 10, maximum 值为 60, 因此滚动轴一开始在刻度 10 的位置上, 因为 minimum 值设为 10, 可滚动的区域大小为 $60-20-10=30$ 个刻度, 滚动范围在 10~40 中。程序第 31 行, 当鼠标在滚动轴列上按一下时, 滚动轴一次所跳的区块大小为 5 个刻度。
- (7) 程序第 48~54 行, 实现 adjustmentValueChanged() 方法。当用户改变转轴位置时, 会将目前的滚动轴刻度写在 label2 上。程序第 51 行, e.getValue() 所得的值与 scrollbar1.getValue() 所得的值是一样的。

⊕ 程序运行结果如图 5-22 所示。



图 5-22

读者若亲自运行这个程序可以发现, 当拖曳转轴时图形并不会跟着转动, 这是因为 JScrollBar 本身没有 JViewport 的机制, 它只管滚动轴滚动的部分。因此若要使图形跟着滚动轴滚动, 就必须自行写其他程序代码来做控制, 而通常我们不会这么做, 因为利用 ScrollPane 就可以帮我们把这些事都处理掉了, 因此用户若想在窗口上有滚动的效果, 建议直接使用 JScrollPane, 然后再对 JScrollPane 上的 JScrollBar 做其他设置, 这是最方便的做法。

5-9 本章总结

Frame 与 Panel 是我们放置组件最常用到的容器, 在本章中我们详细地介绍了这些容器的使用方法, 使用时机等等。Swing 的容器结构有别于旧的 AWT 结构, 如利用 JLayeredPane

可以让用户很方便地控制组件间的层次关系。Internal Frame 可以让用户在一个 Frame 内再自行拥有自己的子 Frame，Internal Frame 由 DesktopPane 来管理。SplitPane 可将窗口切割成两部分，可做出一般网页中切割窗口的效果。JTabbedPane 可让用户做出文件夹的效果，方便文件或信息的管理。JScrollPane 与 JScrollBar 可让用户利用转轴转动窗口，以便看到完整的组件内容。当您读完此章节时，相信您会对版面的设计有一个很完整的概念。

5-10 本章习题

1. 试说明 Swing Top-Level 的容器结构。
2. 试说明 Z-order 中 Layer 与 Position 数值的关系。
3. 实现一个 JLayeredPane 的程序，版面上有两个按钮，一个写“往上一层”，一个写“往下一层”，当用户按这两个按钮时就可以控制组件间的层次关系。
4. 更改 JTabbedPane1.java 程序，使它具有删除标签的功能。
5. 试说明 JScrollPane 的结构关系，并简单实践。

6

标签与按钮的使用与介绍

JLabel 与 JButton 是经常被使用到的 Swing 组件，以往要在 Label 或 Button 中加入图像或其他效果是非常困难的一件事。在 Swing 出来之后，一切困难都迎刃而解了。不仅解决了这些问题，而且在使用上相当的方便。若读者有使用过 AWT 组件的经验，一定能深刻体验到 Swing 为 Java 用户界面带来的冲击。在本章中一开始我们先介绍 Border 与 Icon 类的使用，让用户在使用 JLabel、JButton 或其他组件上有更多的变化！

Swing 组件相当丰富, 足够应付用户多样的需求。在本章中, 我们将先针对用户最常使用的两个 Swing 组件来做介绍, 一个是 JLabel (标签) 组件, 另一个则是 JButton (按钮) 组件。在介绍这两个组件之前, 我们先介绍在 Swing 组件中经常使用的两个类: Border 与 Icon。这两个类也经常使用在 JLabel 与 JButton 上, 帮助设计者制作更美观的用户界面。

6-1 Border 的使用

Border 类是应用在描绘组件的边界, Border 本身是一个 Interface, 里面定义了 3 种方法, 为 `getBorderInsets()`、`isBorderOpaque()`、与 `paintBorder()`。若您想使用 Border 类来绘制您的窗口边界, 您必须先实现 (implements) 这 3 种方法, 可说是有点麻烦。还好, Java 本身提供了另一个类, 它已经实现了 Border 类所有的方法, 且提供许多不同的边界样式供用户使用, 用户只需要选择到底要用哪个样式即可, 不需要理会如何去画出这个边界, 因为这个类已经都帮您实现好了, 这个类就是 `BorderFactory`, 下面是 `BorderFactory` 的类层次结构图。

`BorderFactory` 的类层次结构图:

```
java.lang.Object
    --javax.swing.BorderFactory
```

在 `BorderFactory` 类中提供了许多现有的 Border 组件可供用户直接使用, 非常方便。想要在一般组件上绘制边界, 可以使用 `setBorder()` 这个方法, 然后利用 `BorderFactory` 类选择出想要的边界模式, 就可以很轻松地画出组件边界。我们直接来看下面这个简单的例子吧:

范例 SimpleBorder.java (文件位于随书光盘目录 exam\ch6\SimpleBorder.java)

```
1  import javax.swing.border.*;
2  import javax.swing.*;
3  import java.awt.*;
4  import java.awt.event.*;
5
6  public class SimpleBorder{
7
8      public static void main(String arg[]){
9
10         JFrame f = new JFrame("SimpleBorder");
11         Container content = f.getContentPane();
12         JButton b = new JButton();
13         b.setBorder(BorderFactory.createLineBorder(Color.blue,10));
14         content.add(b);
15         f.setSize(200,150);
16         f.show();
17
18         f.addWindowListener(new WindowAdapter() {
19             public void windowClosing(WindowEvent e) {
20                 System.exit(0);
21             }
22         });
23     }
24 }
```

说明：

- (1) 在程序第 13 行中，我们在 JButton b 中放入一个 Border 组件，此 Border 组件是由 BorderFactory 类所提供。BorderFactory 有一个 createLineBorder() 的方法，可以指定边界的颜色与宽度，在此我们指定边界的颜色为蓝色，宽度为 10。
- (2) 第 18~22 行是处理关闭窗口的操作，若您没写这一段，就算您已经关闭了窗口，但程序并不会就此终止。

图 6-1 为此程序的运行结果。



图 6-1

通过此程序的介绍，读者可发现使用 Border 是件非常容易的事，只要清楚知道 BorderFactory 提供哪些 Border 可以使用，就能制作出想要的边界出来。那么现在我们赶紧来看看 BorderFactory 到底提供了哪些 Border。我们由 BorderFactory 所提供的方法，就能得知共有哪些 Border 可以使用，如表 6-1 所示。

表 6-1

返回类型 BorderFactory 方法	
Static Border	createBevelBorder(int type) 建立一个立体的边界，并由参数 type 指定为凹陷或突起，type 可为 BevelBorder.LOWERED 表示凹陷，或是 BevelBorder.RAISED 表示突起
Static Border	createBevelBorder(int type, Color highlight, Color shadow) 建立一个立体的边界，并指定突边与阴影的颜色
Static Border	createBevelBorder(int type, Color highlightOuter, Color highlightInner, Color shadowOuter, Color shadowInner) 建立一个立体的边界，并指定内外部的突边与阴影的颜色
Static CompoundBorder	createCompoundBorder() 建立一个复合边界
Static CompoundBorder	createCompoundBorder(Border outsideBorder, Border insideBorder) 建立一个复合边界，并指定它的内外边界
Static Border	createEmptyBorder() 建立一个空的边界
Static Border	createEmptyBorder(int top, int left, int bottom, int right) 建立一个空的边界，并指定上下左右的宽度，在这些宽度中不能作绘图的效果
Static Border	createEtchedBorder() 建立一个四周有凹痕的边界
Static Border	createEtchedBorder(Color highlight, Color shadow) 建立一个四周有凹痕的边界，并指定突边与阴影的颜色

续上表

返回类型 BorderLayout 方法	
Static Border	createLineBorder(Color color) 建立一个线条边界, 并指定线条的颜色
Static Border	createLineBorder(Color color, int thickness) 建立一个线条边界, 并指定线条的颜色与宽度
Static Border	createLoweredBevelBorder() 建立一个具有凹陷效果的边界, 意义与 createBevelBorder(BevelBorder.LOWERED) 相同
Static MatteBorder	createMatteBorder(int top, int left, int bottom, int right, Color color) 建立一个垫子边界, 这跟 createEmptyBorder 有点像, 但可以指定边界颜色
Static MatteBorder	createMatteBorder(int top, int left, int bottom, int right, Icon tileIcon) 建立一个垫子边界, 并指定边界的花纹
Static Border	createRaisedBevelBorder() 建立一个具有突起效果的边界, 意义与 createBevelBorder(BevelBorder.RAISED) 相同
Static TitledBorder	createTitledBorder(Border border) 建立一个标题边界, 但没有标题名称
Static TitledBorder	createTitledBorder(Border border, String title) 建立一个标题边界, 并指定标题名称, 标题默认位置是 TitledBorder.DEFAULT_JUSTIFICATION 与 TitledBorder.DEFAULT_POSITION, 也就是左上方
Static TitledBorder	createTitledBorder(Border border, String title, int titleJustification, int titlePosition) 建立一个标题边界, 并指定标题名称与标题位置, 参数 titleJustification 可为: <ul style="list-style-type: none"> ● TitledBorder.LEFT ● TitledBorder.CENTER ● TitledBorder.RIGHT ● TitledBorder.DEFAULT_JUSTIFICATION (leading)
	参数 titlePosition 可为: <ul style="list-style-type: none"> ● TitledBorder.ABOVE_TOP ● TitledBorder.TOP (sitting on the top line) ● TitledBorder.BELOW_TOP ● TitledBorder.ABOVE_BOTTOM ● TitledBorder.BOTTOM (sitting on the bottom line) ● TitledBorder.BELOW_BOTTOM ● TitledBorder.DEFAULT_POSITION (top)
Static TitledBorder	createTitledBorder(Border border, String title, int titleJustification, int titlePosition, Font titleFont) 建立一个标题边界, 并指定标题名称、标题位置与字体
Static TitledBorder	createTitledBorder(Border border, String title, int titleJustification, int titlePosition, Font titleFont, Color titleColor) 建立一个标题边界, 并指定标题名称、标题位置、字体与标题颜色
Static TitledBorder	createTitledBorder(String title) 建立一个标题边界, 并指定标题名称, 其他为默认值

我们一个个来看这些 Border 的功能与介绍吧！

BevelBorder: 可制作出具有三维效果的边界。我们来看下面这个例子:

范例 BorderDemo.java (文件位于随书光盘目录 exam\ch6\BorderDemo.java)

```
1  import javax.swing.border.*;
2  import javax.swing.*;
3  import java.awt.*;
4  import java.awt.event.*;
5
6  public class BorderDemo{
7
8      public static void main(String arg[]){
9
10         JFrame f = new JFrame("BorderDemo");
11         Container content = f.getContentPane();
12         JLabel label = new JLabel();
13
14         label.setBorder(BorderFactory.createBevelBorder(BevelBorder.LOWERED));
15         content.add(label);
16         f.setSize(200,150);
17         f.show();
18
19         f.addWindowListener(new WindowAdapter() {
20             public void windowClosing(WindowEvent e) {
21                 System.exit(0);
22             }
23         });
24     }
25 }
26
```

⊕ **说明:**

- (1) 程序第 14~15 行, 我们产生一个 Bevel 的 Border, 并指定此 BevelBorder 是凹陷的 (BevelBorder.LOWERED)。
- (2) 第 19~23 行是处理关闭窗口的操作, 若您没写这一段, 就算您已经关闭了窗口, 但程序并不会就此终止。

⊕ 图 6-2 为此程序的运行结果:



图 6-2

这个功能跟使用 createLoweredBevelBorder()是一样的。若我们将范例程序 BorderDemo.java 的第 14~15 行改成 label.setBorder(BorderFactory.createBevelBorder(BevelBorder.RAISED)); 则运

行结果是边界将会有突起立体的效果（您也可以使用 `createRaisedBevelBorder()` 来达到相同的效果），如图 6-3 所示。



图 6-3

若您想指定亮边与阴影的颜色，您可以将范例程序 `BorderDemo.java` 的第 14~15 行改成 `label.setBorder(BorderFactory.createBevelBorder(BevelBorder.LOWERED,Color.red,Color.blue));`

④ 运行结果将如图 6-4 所示。



图 6-4

或是改成 `label.setBorder(BorderFactory.createBevelBorder(BevelBorder.LOWERED,Color.red,Color.blue,Color.yellow,Color.green));`

④ 运行结果将如图 6-5 所示。



图 6-5

EmptyBorder: 建立一个空的边界，可指定边界的宽度，这在区隔组件之间的距离时可能会用到。

EtchedBorder: 建立一个四周有凹痕的边界，也可以指定突边与阴影的颜色，若我们将范例程序 `BorderDemo.java` 的第 14~15 行改成 `label.setBorder(BorderFactory.createEtchedBorder());`

✎ 运行结果将如图 6-6 所示。



图 6-6

或是改成 `label.setBorder(BorderFactory.createEtchedBorder(Color.red,Color.blue));`
则运行结果将如图 6-7 所示。



图 6-7

LineBorder: 建立一个线条边界, 并可以指定线条的颜色与宽度。若我们将范例程序 `BorderDemo.java` 的第 14~15 行改成 `label.setBorder(BorderFactory.createLineBorder(Color.blue,5));`

运行结果将如图 6-8 所示。



图 6-8

MatteBorder: 建立一个 Matte 边界, 这个方法与 `createEmptyBorder()` 有点像, 但可以指定边界颜色或是利用 Icon 产生边界花纹。若我们将范例程序 `BorderDemo.java` 的第 14~15 行改成 `label.setBorder(BorderFactory.createMatteBorder(5,5,5,5,Color.green));`

✎ 运行结果将如图 6-9 所示。



图 6-9

或是改成 `label.setBorder(BorderFactory.createMatteBorder(25,25,25,25,new ImageIcon ("\\icons\\star.gif")));`

`ImageIcon` 是制作 `Icon` 的类，我们会在下节讨论到。运行结果将如图 6-10 所示。



图 6-10

CompoundBorder: 建立一个复合边界，并可以指定它的内外边界。例如我们可以指定它的外边界为 `LineBorder`，它的内边界为 `MatteBorder`，我们一样更改范例程序 `BorderDemo.java` 的第 14~15 行，改成 `label.setBorder(BorderFactory.createCompoundBorder(BorderFactory.createLineBorder(Color.blue,5),BorderFactory.createMatteBorder(20,20,18,18,new ImageIcon ("\\icons\\star.gif"))));`

运行结果将如图 6-11 所示。



图 6-11

TitleBorder: 建立一个标题边界，我们可以指定边界的标题名称、标题位置、字体与标题颜色。例如我们建立一个 `LineBorder`，然后将此 `LineBorder` 放入 `TitleBorder` 中，并指定标题为“LineBorder”，标题的位置在边界的上缘中间部分。则我们将范例程序 `BorderDemo.java` 的第 14~15 行改成 `label.setBorder(BorderFactory.createTitledBorder(BorderFactory.create LineBorder (Color.blue,5),"Line Border",TitledBorder.CENTER,TitledBorder.TOP));`

运行结果将如图 6-12 所示。



图 6-12

或是改成 `label.setBorder(BorderFactory.createTitledBorder(BorderFactory.createLineBorder (Color.blue,5),"Line Border",TitledBorder.LEFT,TitledBorder.ABOVE_TOP,new Font("SansSerif", Font.ITALIC,14),Color.red));`

则运行结果将如图 6-13 所示。



图 6-13

6-2 Icon 的使用

在以往 AWT 的结构中，想要在一般按钮上加上图形几乎是不可能的事；但在目前 Swing 的结构下，我们可以很轻易地在任何组件上加入 Icon 组件，更丰富了组件的表现效果。

制作 Icon 我们可以利用上节所使用的 ImageIcon 类，这个类可以让我们很轻易地制作出 Icon 组件。ImageIcon 类是实现 Icon 界面而来，Icon 界面里面定义了 3 种方法，分别是 getIconHeight()、getIconWidth()、与 paintIcon()。若您直接使用 ImageIcon 类来制作 Icon 组件，您就无须自行实现这 3 种方法，可说是相当方便。下面是 ImageIcon 的类层次结构图。

ImageIcon 的类层次结构图：

```
java.lang.Object
    -- javax.swing.ImageIcon
```

ImageIcon 提供了相当多的构造函数帮助您构造 Icon 组件，我们稍微来看一下，如表 6-2 所示。

表 6-2

ImageIcon 的构造函数

ImageIcon()	建立一个 ImageIcon 组件
ImageIcon(byte[] imageData)	建立一个 ImageIcon 组件，Image 的数据放在 byte array 中，而 byte array 的数据可从 GIF 获释 JPEG 的文件读取而来
ImageIcon(byte[] imageData, String description)	建立一个 ImageIcon 组件，Image 的数据放在 byte array 中，而 byte array 的数据可从 GIF 获释 JPEG 的文件读取而来，并多了一个此 Icon 的描述文字
ImageIcon(Image image)	利用 Image 组件 建立一个 ImageIcon 组件
ImageIcon(Image image, String description)	利用 Image 组件 建立一个 ImageIcon 组件，并附加此 Icon 的描述文字
ImageIcon(String filename)	利用图文件建立一个 ImageIcon 组件
ImageIcon(String filename, String description)	利用图文件建立一个 ImageIcon 组件，并附加此 Icon 的描述文字
ImageIcon(URL location)	利用网址建立一个 ImageIcon 组件
ImageIcon(URL location, String description)	利用网址建立一个 ImageIcon 组件，并附加此 Icon 的描述文字

利用 ImageIcon 建立 Icon 组件相当简单,例如您已经有一个图文件叫做 visa.gif,若您想把此 Icon 图案放进一个 JLabel 中,您可以利用下面的程序帮您完成:

范例 ImageIconDemo.java (文件位于随书光盘目录 exam\ch6\IconDemo.java)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class ImageIconDemo
6  {
7      public static void main(String args[])
8      {
9          JFrame f = new JFrame("ImageIconDemo");
10         Container contentPane = f.getContentPane();
11         Icon icon = new ImageIcon(".\\icons\\visa.gif");
12         JLabel label = new JLabel(icon, JLabel.CENTER);
13         contentPane.add(label);
14         f.pack();
15         f.show();
16
17         f.addWindowListener(new WindowAdapter() {
18             public void windowClosing(WindowEvent e) {
19                 System.exit(0);
20             }
21         });
22     }
23 }
```

⊕ 说明:

- (1) 程序第 11 行中,我们利用 ImageIcon 类制作出一个 Icon 组件。
- (2) 程序第 12 行中,将 Icon 组件放入 JLabel 中,并设置其位置在 JLabel 的中间。

⊕ 程序运行结果如图 6-14 所示。



图 6-14

若您想利用读文件的功能,应先将图文件转成 byte array,再制作 Icon 组件,您可以用下面的程序帮您完成:

范例 ImageIconDemo1.java (文件位于随书光盘目录 exam\ch6\IconDemo1.java)

```
1  import java.awt.*;
2  import java.awt.event.*;
```

```

3    import javax.swing.*;
4    import java.io.*;
5
6    public class ImageIconDemo1
7    {
8        public static void main(String args[])
9        {
10            byte[] image = null;
11
12            JFrame f = new JFrame("ImageIconDemo");
13            Container contentPane = f.getContentPane();
14
15            try{
16                File file = new File(".\\icons\\visa.gif");
17                int size = (int)file.length();
18                FileInputStream in = new FileInputStream(file);
19                image = new byte[size];
20                in.read(image);
21            } catch (IOException e) {
22                System.err.println("File open failure:"+e.getMessage());
23            }
24
25            Icon icon = new ImageIcon(image);
26            JLabel label = new JLabel(icon, JLabel.CENTER);
27            contentPane.add(label);
28            f.pack();
29            f.show();
30            f.addWindowListener(new WindowAdapter() {
31                public void windowClosing(WindowEvent e) {
32                    System.exit(0);
33                }
34            });
35        }
36    }

```

⊕ 说明:

- (1) 在打开文件读取的过程中可能会发生 `IOException`, 因此在第 15 与第 21 行中, 用 `try` 与 `catch` 将此区段包起来。
- (2) 程序第 16~17 行, 利用文件 `visa.gif` 建立一个 `File` 组件, 并求出此文件的长度。
- (3) 程序第 18~20 行, 将文件组件放入 `FileInputStream` 中, 并利用 `FileInputStream` 中的 `read()` 方法, 将文件的数据读进 `byte array` 中。
- (4) 程序第 25 行, 利用 `byte array` 的图文件数据建立 `Icon` 组件。
- (5) 程序第 26 行, 将 `Icon` 组件放入 `JLabel` 中, 并置于中央。

程序运行结果将会跟上个范例一模一样, 读者不妨试试看。

以上我们所看到的均是利用 `ImageIcon` 类来建立 `Icon` 组件, 建立的过程相当简单。若我们想利用 Java 的绘图功能, 绘制出一个图形, 那我们可以直接利用 `Icon` 类来建立 `Icon` 组件吗? 答案当然是肯定的, 但如我们以前所说, `Icon` 它是一个 `Interface`, 里面包含 3 种空的抽象方法, 分别是 `getIconHeight()`、`getIconWidth()` 与 `paintIcon()`; 若您想使用 `Icon` 类来制作 `Icon`,

您必须实现(implements)这 3 种方法, 我们直接来看下面这个例子, 您就能明白如何实现 Icon 界面建立 Icon 组件了。

范例 IconDemo.java (文件位于随书光盘目录 exam\ch6\IconDemo.java)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class IconDemo implements Icon
6  {
7      int height = 50;
8      int width = 70;
9
10     public int getIconHeight()
11     {
12         return height;
13     }
14
15     public int getIconWidth()
16     {
17         return width;
18     }
19
20     public void paintIcon(Component c, Graphics g, int x, int y)
21     {
22         g.drawRect(x, y, width, height);
23         g.fillRect(x, y, width, height);
24         g.setColor(Color.blue);
25     }
26
27     public static void main(String args[])
28     {
29         JFrame f = new JFrame("IconDemo");
30         Container contentPane = f.getContentPane();
31
32         Icon icon = new IconDemo();
33         JLabel label = new JLabel(icon, JLabel.CENTER);
34         contentPane.add(label);
35         f.pack();
36         f.show();
37
38         f.addWindowListener(new WindowAdapter() {
39             public void windowClosing(WindowEvent e) {
40                 System.exit(0);
41             }
42         });
43     }
44 }
45
```

◆ 说明:

- (1) Icon 本身是一个 Interface, 因此程序一开始在第 5 行中必须先 implements Icon 界面, 并在下面实现 Icon 界面所定义的 3 种方法。实现方法的程序代码分别写在第

10、15 与 20 行中。

- (2) 在此范例中，我们的目的是要用 `Icon` 类产生一个蓝色矩形图标的 `Icon` 组件，在程序第 7~8 行中，我们先声明此矩形图标的大小，程序第 10~18 行实现可返回矩形高与宽的方法，而真正将矩形图标画出来是在程序第 20~25 行中。
- (3) 程序第 32 行，我们利用 `IconDemo` 类产生一个 `Icon` 组件，并在程序第 33 行中，将此 `Icon` 组件放入 `JLabel` 中。
- (4) 当程序运行到第 36 行时，系统会自动调用 `paintIcon()` 方法，并自动传入 `Component` 与 `Graphics` 参数，将蓝色矩形图标画出来。

④ 程序运行结果如图 6-15 所示。



图 6-15

6-3 JLabel 的使用

类层次结构图：

```
java.lang.Object
    --java.awt.Component
    --java.awt.Container
        --javax.swing.JComponent
            --javax.swing.JLabel
```

在这以前的许多范例中，我们已经使用过 `JLabel` 这个组件，相信大家对此组件应该不会感到陌生。接下来我们将深入了解 `JLabel` 的各种特性。

一般而言，我们最常在 `JLabel` 上放置文字或图形，也因此我们经常需要调整 `JLabel` 上文字或图形的相关位置。在 Java 中，`JLabel` 实现了 `SwingConstants` 这个 `Interface`，而这个 `Interface` 主要是定义一些组件排列方式的参数，如表 6-3 所示。

表 6-3

TOP	置于顶端
LEFT	置于左边
RIGHT	置于右边
BOTTOM	置于下端
CENTER	置于中间
NORTH	置于北边
EAST	置于东边

续上表

WEST	置于西边
SOUTH	置于南边
NORTH_EAST	置于东北边
SOUTH_EAST	置于东南边
SOUTH_WEST	置于西南边
NORTH_WEST	置于西北边
HORIZONTAL	水平排列
VERTICAL	垂直排列
LEADING	置于前端
TRAILING	置于后端

在 Swing 中,有相当多的类均实现了 `SwingConstants` 这个 Interface,如 `AbstractButton`、`JCheckBoxMenuItem`、`JLabel`、`JProgressBar`、`JSeparator`、`JSlider`、`TextField`、`JTabbedPane`、`JToolBar` 等等,因此当您使用到这些组件时,您就可以在适当的时候,利用 `SwingConstants` 的参数来定义组件的位置了。

`JLabel` 共有 6 种构造函数,我们来了解一下,以便在使用时更为方便,如表 6-4 所示。

表 6-4

JLabel 的构造函数	
<code>JLabel()</code>	建立一个空白的 <code>JLabel</code> 组件
<code>JLabel(Icon image)</code>	建立一个含有 <code>Icon</code> 的 <code>JLabel</code> 组件, <code>Icon</code> 的默认排列方式是 <code>CENTER</code>
<code>JLabel(Icon image, int horizontalAlignment)</code>	建立一个含有 <code>Icon</code> 的 <code>JLabel</code> 组件,并指定其排列方式
<code>JLabel(String text)</code>	建立一个含有文字的 <code>JLabel</code> 组件,文字的默认排列方式是 <code>LEFT</code>
<code>JLabel(String text, int horizontalAlignment)</code>	建立一个含有文字的 <code>JLabel</code> 组件,并指定其排列方式
<code>JLabel(String text, Icon icon, int horizontalAlignment)</code>	建立一个含有文字与 <code>Icon</code> 的 <code>JLabel</code> 组件,并指定其排列方式,文字与 <code>Icon</code> 的间距,默认值是 4 个 pixels

在 `JLabel` 中,有几种方法可能是常用到的,例如 `setHorizontalAlignment(int alignment)` 与 `setVerticalAlignment(int alignment)`,分别是设置标签内组件(文字或 `Icon`)的水平或垂直位置,而 `setHorizontalTextPosition(int textPosition)` 与 `setVerticalTextPosition(int textPosition)` 可设置文字相对于 `Icon` 的相对位置,另外, `setIconTextGap(int iconTextGap)` 可设置标签内文字与 `Icon` 间的间距、`setText(String test)` 与 `setIcon(Icon icon)` 可分别设置标签内的文字与 `Icon`。

我们来看几个 `JLabel` 的例子吧。

下面这个范例是设置标签内文字的排列方式:

范例 JLabelDemo1.java (文件位于随书光盘目录 exam\ch6\ JLabelDemo1.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JLabelDemo1
6  {
7      public static void main(String args[])
8      {
9          JFrame f = new JFrame("JLabelDemo1");
10         Container contentPane = f.getContentPane();
11         JLabel label = new JLabel();
12         label.setText("Hello");
13         label.setHorizontalAlignment(JLabel.RIGHT);
14         label.setVerticalAlignment(JLabel.TOP);
15         contentPane.add(label);
16         f.pack();
17         f.show();
18         f.addWindowListener(new WindowAdapter() {
19             public void windowClosing(WindowEvent e) {
20                 System.exit(0);
21             }
22         });
23     }
24 }

```

说明：

- (1) 在程序第 11 行，我们产生一个空白的 JLabel 组件。
- (2) 程序第 12~14 行，设置 JLabel 的文字为“Hello”，并设置“Hello”的位置在 JLabel 中的右上角。
- (3) 其实我们可以将程序第 11~13 行缩减成 JLabel label = new JLabel(“Hello”, JLabel.RIGHT)。

程序运行结果如图 6-16 所示。

图 6-16

我们再来看一个有 Icon 的例子：

范例 JLabelDemo2.java (文件位于随书光盘目录 exam\ch6\ JLabelDemo2.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4

```

```
5 public class JLabelDemo2
6 {
7     public static void main(String args[])
8     {
9         JFrame f = new JFrame("JLabelDemo2");
10        Container contentPane = f.getContentPane();
11        Icon icon = new ImageIcon(".\\icons\\hello.jpg");
12        JLabel label = new JLabel("Hello", icon, JLabel.CENTER);
13        label.setHorizontalTextPosition(JLabel.CENTER);
14        label.setVerticalTextPosition(JLabel.TOP);
15        contentPane.add(label);
16        f.pack();
17        f.show();
18        f.addWindowListener(new WindowAdapter() {
19            public void windowClosing(WindowEvent e) {
20                System.exit(0);
21            }
22        });
23    }
24 }
```

⊕ 说明:

- (1) 程序第 11 行, 利用 ImageIcon 类将 “hello.jpg” 文件生成一个 Icon 组件。
- (2) 程序第 12 行, 产生一个具有文字与 Icon 的 JLabel 组件, 并将此文字与 Icon 置于 JLabel 的中间。
- (3) 程序第 13 行, 将文字置于 Icon 的中间, 若没有设置此项, 默认值为文字在 Icon 的右边。
- (4) 程序第 14 行, 将文字置于 Icon 的上面, 若没有设置此项, 默认值为中间排列。

⊕ 程序运行结果如图 6-17 所示。



图 6-17

若您将程序第 13~14 行给 Mark 起来 (当成注释), 则程序运行结果将如图 6-18 所示。



图 6-18

若您在上面程序中再加入一行 `label.setIconTextGap(10);` 将会加大文字“Hello”与 Icon 间的间距, 如图 6-19 所示。



图 6-19

6-4 JButton 的使用

JButton 的类层次结构图:

```
java.lang.Object
    --java.awt.Component
        --java.awt.Container
            --javax.swing.JComponent
                --javax.swing.AbstractButton
                    --javax.swing.JButton
```

JButton 与 JLabel 都是经常被使用到的 Swing 组件, 由上面的层次结构图中, 我们可以很清楚地知道 JButton 是继承 AbstractButton 类而来, 而 AbstractButton 本身是一个抽象类, 里面定义了许多组件设置的方法与组件事件驱动的方法 (Event Handle), 如 `addActionListener()`、`setText()`、`setRolloverIcon()`、`setHorizontalAlignment()`、`setEnabled()`、`isSelected()` 等等, 所提供的方法不下 50 种, 可说是非常重要的一个类。事实上, AbstractButton 类不仅被 JButton 所继承, 它同时还被 JMenuItem、JToggleButton、JCheckBox、JRadioButton 等类所继承, 提供给这些类强大且方便的功能, 而且在使用上更能掌握这些组件的特性。我们在此节中先来了解 JButton 与 JToggleButton 的特性, 其余类将在后面各章节中介绍。

JButton 共有 4 个构造函数, 我们先来了解一下, 如表 6-5 所示。

表 6-5

JButton 的构造函数

<code>JButton()</code>	建立一个按钮
<code>JButton(Icon icon)</code>	建立一个有图像的按钮
<code>JButton(String text)</code>	建立一个有文字的按钮
<code>JButton(String text, Icon icon)</code>	建立一个有图像与文字的按钮

由 JButton 的构造函数可以看出, JButton 与 JLabel 的使用相当类似, 只是 JButton 少了排列方式的参数罢了。要是我们想设置 JButton 内文字或图像的水平排列方式, 我们可以利用 AbstractButton 抽象类所提供的 `setHorizontalAlignment()` 方法来达成。JButton 在使用上与

JLabel 相当类似，只是类的设计方式有所不同；JLabel 类自行提供组件排列方式的方法，而 JButton 是继承 AbstractButton 抽象类的方法来排列按钮内的内容。为什么 JButton 不自行提供排列方式等方法呢？主要是因为 JButton 与 JMenuItem、JToggleButton、JCheckBox、JRadioButton 组件有许多共同的特性，例如它们都会有“按”的操作、都可以插入 Icon 与文字、都可设置快捷键、都可呈现 Enable 或 Disable 状态等等，因此将 AbstractButton 类独立出来，实现这些共通的方法，再由其他类来继承，将可增强程序结构对象化与模块化的特性，也让程序更容易维护（其实读者有兴趣可以偶尔看看 Java 本身的设计方式，这会让您大大的增加对象化程序写作的功力喔）。

JButton 类所提供的方法非常少，大部分都会用到 AbstractButton 抽象类所提供的方法。我们一起来看看下面这几个例子，就能知道 JButton 常用的功能有哪些，以及如何使用。

范例 JButtonDemo1.java (文件位于随书光盘目录 exam\ch6\JButtonDemo1.java)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JButtonDemo1
6  {
7      public static void main(String args[])
8      {
9          JFrame f = new JFrame("JButtonDemo1");
10         Container contentPane = f.getContentPane();
11         JButton b = new JButton("按我", new ImageIcon("./icons\\hand.jpg"));
12         b.setHorizontalTextPosition(JButton.CENTER);
13         b.setVerticalTextPosition(JButton.BOTTOM);
14         contentPane.add(b);
15         f.pack();
16         f.show();
17         f.addWindowListener(new WindowAdapter() {
18             public void windowClosing(WindowEvent e) {
19                 System.exit(0);
20             }
21         });
22     }
23 }
```

⊕ 说明：

- (1) 首先在程序第 11 行中先产生一个按钮，这个按钮包括了“按我”的文字，与一个 Icon 图文件。
- (2) 程序第 11 行您也可以用下面这个方式代替：

```
JButton b = new JButton();
b.setIcon(new ImageIcon("./icons\\hand.jpg"));
b.setText("按我");
```
- (3) 程序第 12~13 行，我们将“按我”文字置于图形下面的中间部分。若您没有设置文字的位置，系统默认会将文字置于图形的右边中间位置。
- (4) 程序第 17~21 行，处理窗口关闭的操作。

④ 程序运行结果如图 6-20 所示。



图 6-20

6-4-1 在 JButton 上使用 Rollover 图像变化

您如果是位经常上网的人，应该常会看到许多网页有这样的效果：那就是当光标放在按钮上与光标离开按钮时，其按钮上会显示出不同的图形效果。我们利用 Java 可以很容易就实现这个功能，赶紧来看下面这个范例吧：

范例 RolloverDemo.java (文件位于随书光盘目录 exam\ch6\RolloverDemo.java)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class RolloverDemo
6  {
7      public static void main(String args[])
8      {
9          JFrame f = new JFrame("RolloverDemo");
10         Container contentPane = f.getContentPane();
11         Icon rollover = new ImageIcon(".\\icons\\address1.jpg");
12         Icon general = new ImageIcon(".\\icons\\address2.jpg");
13         Icon press = new ImageIcon(".\\icons\\address3.jpg");
14         JButton b = new JButton();
15         b.setRolloverEnabled(true);
16         b.setIcon(general);
17         b.setRolloverIcon(rollover);
18         b.setPressedIcon(press);
19         contentPane.add(b);
20         f.pack();
21         f.show();
22         f.addWindowListener(new WindowAdapter() {
23             public void windowClosing(WindowEvent e) {
24                 System.exit(0);
25             }
26         });
27     }
28 }
```

⊕ 说明:

- (1) 在此程序中我们将其设计成当鼠标指针在按钮上时, 会显示出 address1.jpg 的图形出来, 当鼠标按下按钮时会显示出 address3.jpg 图形, 当鼠标离开按钮时则显示出 address2.jpg 图形出来。
- (2) 程序第 14 行, 我们先产生一个按钮组件。
- (3) 程序第 15 行, 我们将 Rollover 的功能打开。
- (4) 程序第 16~18 行, 我们分别设置鼠标离开按钮, 在按钮上, 按下按钮时的图像。

⊕ 程序运行结果如下:

当鼠标不在按钮上时, 如图 6-21 所示。



图 6-21

当鼠标在按钮上时, 如图 6-22 所示。



图 6-22

当鼠标按下按钮时, 如图 6-23 所示。



图 6-23

除了各位看到的这些功能外, `AbstractButton` 抽象类也提供了 `setDisabledIcon()`、`setDisableSelectedIcon()`、`setSelectedIcon()`、`setRolloverSelectedIcon()` 等功能, 其中后三项是用在 `ToggleButton` 中, 我们会在下节中介绍。读者可以自行试试 `setDisabledIcon()` 方法, 我们提供 Disabled Icon 的图形为 address4.jpg, 看运行的结果是否会如图 6-24 所示呢!



图 6-24

6-4-2 在 JButton 上设置快捷键

在使用某些软件时，我们常常会发现这些软件均具有快捷键的功能，所谓快捷键就是您不需要将光标移到该按钮或是选项上，只需要按下快捷键所设置的英文字母，就能运行该选项所提供的功能，这对已经习惯使用键盘的人来说无疑是一项非常方便的工具。而在 Java 的按钮或菜单中，也提供了这样的功能，而且非常容易使用。在下面这个例子中，除了使用到快捷键的功能外，我们还使用到事件处理（Event Handle）的方法。

在 Java 中，所有的组件都有他相对应的事件处理方式，例如按下按钮后新增一个窗口、选择菜单（Menu）里的某个选项后运行某个方法等，在上一章节中我们已经深入介绍了事件的原理与处理方法，在此我们也将开始介绍各种组件的事件处理模式。因此读者若能将各种 Swing 组件熟悉之后，并了解一些常用的事件处理模式，再来深入探讨所有事件处理的方法，便能构造出多功能的用户界面，并能灵活的运用各种 Swing 组件的互动性。

接下来，我们赶紧来看如何在 Java 的按钮上设置快捷键吧：

范例 MnemonicButton.java (文件位于随书光盘目录 exam\ch6\ MnemonicButton.java)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class MnemonicButton implements ActionListener
6  {
7      public MnemonicButton()
8      {
9          JFrame f = new JFrame("MnemonicButton");
10         Container contentPane = f.getContentPane();
11         JButton b = new JButton("Open new window");
12         b.setMnemonic('O');
13         b.addActionListener(this);
14         contentPane.add(b);
15         f.pack();
16         f.show();
17         f.addWindowListener(new WindowAdapter() {
18             public void windowClosing(WindowEvent e) {
19                 System.exit(0);
20             }
21         });
22     }
23
24     public void actionPerformed(ActionEvent e)
25     {
```

```
26         JFrame newf = new JFrame("新窗口");
27         JLabel label = new JLabel("这是新窗口");
28         label.setHorizontalAlignment(JLabel.CENTER);
29         newf.getContentPane().add(label);
30         newf.setSize(100,100);
31         newf.show();
32     }
33
34     public static void main(String arg[])
35     {
36         new MnemonicButton();
37     }
38 }
```

⊕ 说明：

- (1) 程序第 11 行，我们先产生一个具有“Open new window”文字的按钮。
- (2) 程序第 12 行，设置快捷键符号为‘O’，则程序在运行时会在“Open new window”按钮中的‘O’处有一个下划线符号，表示快捷键设为‘O’，您只需要按【Alt+O】，无需用到鼠标，就可以运行那按钮所提供的功能。快捷键符号一般都是设按钮文字上的第一个英文字母，当然你可以不这么做，例如若您将快捷键符号设为‘d’，则在“Open new window”按钮中的‘d’处就会有一个下划线符号，按下【Alt+D】一样具有同样的功能；若您将快捷键符号设为‘n’，则下划线的符号会出现在“Open”的 n 上，而不是“new”的 n 上。
- (3) 程序第 13 行，将按钮 b 加入事件处理模式。当我们按下按钮时，会产生一个事件（ActionEvent），此事件会被 ActionListener 所接收。而 ActionListener 类是一个 Interface，里面只有 actionPerformed() 一个方法，因此我们必须实现 actionPerformed() 方法，处理我们所要的结果，即程序的 24~32 行：产生一个新的窗口。
- (4) 在程序一开始的地方，也就是程序的第 5 行，必须写成 implements ActionListener，这样才能将窗口上的按钮加入事件处理模式中，这点非常重要，读者可不要忘记了。若您没有在一开始写 implements ActionListener，那么除非您使用 Inner Class 中匿名类的写法，否则程序在编译时就会有 Error 出现。

⊕ 程序运行结果如图 6-25 所示。



图 6-25

一开始程序会弹出含有一个按钮的窗口，您可以直接使用鼠标来按这个按钮，或利用快捷键功能，按下【Alt+O】就会产生一个新的窗口，如图 6-26 所示。



图 6-26

6-4-3 设置默认按钮

除了我们刚刚看到的快捷键功能外，许多软件也会提供默认按钮的功能，默认按钮的好处是用户无须移动鼠标，也不需要按快捷键，只需要按下【Enter】键就可以运行该默认按钮所提供的功能。例如我们在安装软件时，大部分的软件都会将“下一步”按钮设为默认按钮，用户只需要一直按【Enter】键就能将软件安装成功。在 Java 中要设置默认按钮可以使用 `JRootPane` 类所提供的 `setDefaultButton()` 方法。我们来看个例子吧，在这个例子中我们将事件处理模式换成另一种写法，也就是前面所提到的 Inner Class 匿名类写法，因此在类 `DefaultButton` 的右边就不用再写入 `implements ActionListener` 了。

范例 `DefaultButton.java` (文件位于随书光盘目录 `exam\ch6\DefaultButton.java`)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class DefaultButton
6  {
7      public DefaultButton()
8      {
9          JFrame f = new JFrame("DefaultButton");
10         Container contentPane = f.getContentPane();
11         contentPane.setLayout(new GridLayout(1,2));
12         JButton b1 = new JButton("Open Text window");
13         JButton b2 = new JButton("Open Image window");
14         b1.setMnemonic('T');
15         b2.setMnemonic('I');
16         f.getRootPane().setDefaultButton(b1);
17
18         b1.addActionListener(new ActionListener()
19         {
20             public void actionPerformed(ActionEvent e){
21                 JFrame newf = new JFrame("新窗口");
22                 JLabel label = new JLabel("这是文字窗口");
23                 label.setHorizontalAlignment(JLabel.CENTER);
24                 newf.getContentPane().add(label);
25                 newf.setSize(200,200);
26                 newf.show();
27             }
28         });
29
30         b2.addActionListener(new ActionListener()
31         {

```

```

32         public void actionPerformed(ActionEvent e) {
33             JFrame newf = new JFrame("新窗口");
34             JLabel label = new JLabel(new ImageIcon("..\icons\address-
35                 1.jpg"));
36             label.setHorizontalAlignment(JLabel.CENTER);
37             newf.getContentPane().add(label);
38             newf.setSize(200,200);
39             newf.show();
40         }
41     });
42     contentPane.add(b1);
43     contentPane.add(b2);
44     f.pack();
45     f.show();
46     f.addWindowListener(new WindowAdapter() {
47         public void windowClosing(WindowEvent e) {
48             System.exit(0);
49         }
50     });
51 }
52
53 public static void main(String arg[])
54 {
55     new DefaultButton();
56 }
57 }

```

◆ 说明:

- (1) 程序第 11 行, 设置 contentPane 为 GridLayout(1,2), 一行两列。
- (2) 程序第 12~15 行, 新增两个按钮, 并设置他们的快捷键字母。
- (3) 程序第 16 行, 设置默认按钮是 b1, 也就是“Open Text window”按钮。
- (4) 程序第 18~28 行, 将 b1 的按钮事件加进 ActionListener 中, 并利用 Inner Class 中匿名类方法处理 actionPerformed()。在 actionPerformed()方法中写入产生一个新窗口的程序代码。
- (5) 程序第 30~40 行同第 18~28 行的模式。
- (6) 程序第 42~43 行, 分别将按钮 b1 与 b2 加进 contentPane 中。

◆ 程序运行结果如图 6-27 所示。



图 6-27

由于我们将“Open Text window”设为默认按钮，因此您只要直接按下【Enter】键就可以运行产生一个文字窗口的功能，如图 6-28 所示。



图 6-28

若您想要运行“Open Image window”的功能，您必须使用鼠标来点选，或是利用快捷键【Alt+I】来打开图像窗口的功能，运行结果如图 6-29 所示。



图 6-29

另外，您也可以利用 Tab 键来转换焦点（Focus），然后按下空格键来运行按钮上所提供的方法。读者朋友们不妨试试看。

6-5 JToggleButton 的使用

JToggleButton 类层次结构图：

```
java.lang.Object
    --java.awt.Component
        --java.awt.Container
            --javax.swing.JComponent
                --javax.swing.AbstractButton
                    --javax.swing.JToggleButton
```

JToggleButton 跟一般 JButton 其实很像，主要的差别在于一般的按钮按下去会自动弹回来，而 JToggleButton 按钮按下去只会陷下去，而不会弹回来，除非您再按一次。由上面类层次结构图可知 JToggleButton 也是继承 AbstractButton 抽象类而来，因此 JToggleButton 组件可以使用所有 AbstractButton 类所提供的方法。表 6-6 是 JToggleButton 所提供的构造函数。

表 6-6

JToggleButton 的构造函数

JToggleButton()
建立一个新的 JToggleButton

JToggleButton 的构造函数

JToggleButton(Icon icon)

建立一个有图像但没有文字的 JToggleButton

JToggleButton(Icon icon, boolean selected)

建立一个有图像但没有文字的 JToggleButton, 且设置其初始状态 (有无被选取)

JToggleButton(String text)

建立一个有文字的 JToggleButton

JToggleButton(String text, boolean selected)

建立一个有文字的 JToggleButton, 且设置其初始状态 (有无被选取)

JToggleButton(String text, Icon icon)

建立一个有文字且有图像的 JToggleButton, 初始状态为未被选取

JToggleButton(String text, Icon icon, boolean selected)

建立一个有文字且有图像的 JToggleButton, 且设置其初始状态 (有无被选取)

我们来看下面这个简单的例子:

范例 ToggleButton.java (文件位于随书光盘目录 exam\ch6\ToggleButtonButton.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class ToggleButton
6  {
7      public static void main(String args[])
8      {
9          JFrame f = new JFrame("ToggleButton");
10         Container contentPane = f.getContentPane();
11         contentPane.setLayout(new GridLayout(3,1));
12         JToggleButton b1 = new JToggleButton("Button 1");
13         JToggleButton b2 = new JToggleButton("Button 2");
14         JToggleButton b3 = new JToggleButton("Button 3");
15         contentPane.add(b1);
16         contentPane.add(b2);
17         contentPane.add(b3);
18         f.pack();
19         f.show();
20         f.addWindowListener(new WindowAdapter() {
21             public void windowClosing(WindowEvent e) {
22                 System.exit(0);
23             }
24         });
25     }
26 }

```

说明:

- (1) 程序第 11 行中, 设置 contentPane 为 GridLayout (3, 1), 三行一列。
- (2) 程序第 12~14 行, 产生 3 个 JToggleButton: b1、b2、b3, 并在第 15~17 行中加入 contentPane 中。

◆ 程序运行结果如图 6-30 所示。



图 6-30

当您按下“Button 1”与“Button 3”按钮时，此时“Button 1”与“Button 3”按钮处于下压状态，并不会自动复原，如图 6-31 所示。



图 6-31

您必须再按一次“Button 1”与“Button 3”按钮才会使它们恢复原来的状态。

在 5-5-1 节中，我们曾介绍过如何在 JButton 上使用 Rollover 的方法，且还有 3 种方法我们还没有使用过，那就是 setDisableSelectedIcon()、setSelectedIcon()与 setRolloverSelectedIcon()。我们来简单介绍一下 setSelectedIcon()与 setRolloverSelectedIcon()方法，而 setDisableSelectedIcon()方法就留做习题，读者可自行试用。

范例 ToggleRollover.java (文件位于随书光盘目录 exam\ch6\ ToggleRollover.java)

```
1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4
5 public class ToggleRollover
6 {
7     public static void main(String args[])
8     {
9         JFrame f = new JFrame("ToggleRollover");
10        Container contentPane = f.getContentPane();
11        JToggleButton b = new JToggleButton();
12        b.setRolloverEnabled(true);
13        b.setIcon(new ImageIcon(".\\icons\\rollover1.jpg"));
14        b.setRolloverSelectedIcon(new ImageIcon(".\\icons\\rollover2.jpg"));
```

```
15         b.setSelectedIcon(new ImageIcon(".\\icons\\rollover3.jpg"));
16         contentPane.add(b);
17         f.pack();
18         f.show();
19         f.addWindowListener(new WindowAdapter() {
20             public void windowClosing(WindowEvent e) {
21                 System.exit(0);
22             }
23         });
24     }
25 }
```

说明：

- (1) 程序第 11 行产生一个 `JToggleButton` 组件。
- (2) 程序第 12 行，将 `Rollover` 的功能打开。
- (3) 程序第 13~15 行，分别设置按钮在一般状态、按下状态、按下且略过状态所需显示的图案。

程序运行结果如下：

一开始在未按下按钮时的状态，如图 6-32 所示。



图 6-32

按下按钮后的状态，如图 6-33 所示。



图 6-33

按下按钮且光标在按钮上的时候，如图 6-34 所示。



图 6-34

6-6 本章总结

本章详细地介绍了 `Border`、`BorderFactory`、`Icon`、`ImageIcon`、`JLabel`、`JButton`、`JToggleButton` 的使用方法。例如如何为组件勾画边界、如何制作 `Icon` 组件、如何插入图片到 `JLabel` 与 `JButton` 中、如何对 `JLabel` 与 `JButton` 中的图片与文字做排列、如何制做 `Rollover` 的效果、如何设置默认按钮、如何设置快捷键、如何使用 `JToggleButton` 等等，均是本章节的重点，另外我们也对按钮的事件处理模式作了些说明。希望读者在熟悉这些组件后，对于组件的事件处理也能做到得心应手。

6-7 本章习题

1. 利用 `BorderFactory` 中的 `MatteBorder` 建立一个具有图形边界的 `JLabel` 组件出来。
2. 试实现 `Icon` 类的 `getIconHeight()`、`getIconWidth()` 与 `paintIcon()` 方法，画出一个红色的圆形，然后置入一个 `JButton` 中。
3. 试说明如何排列 `JLabel` 与 `JButton` 中的文字与图像。
4. 试利用 `setDisabledIcon()` 的方法，使得结果如 5-5-1 节中最后一个图案所示。
5. 利用事件处理模式，在窗口上置入一个 `JLabel` 与 `JButton`，当按下 `JButton` 时，`JLabel` 上的文字会跟着改变，如图 6-35 所示。



图 6-35

按下“Change Text”按钮后，`JLabel` 的文字变为如图 6-36 所示。



图 6-36

7

复选框、选项按钮、列表方框、 下拉式列表的使用与介绍

一个功能较强大的软件必定提供许多复杂的选项供用户选取，此时您可以利用本章所介绍的四个组件满足一些相关的需求。若您现在正想知道如何利用 Java 设计一份网络问卷的话，赶快进来看此章内容吧！



7-1 使用 JCheckBox 组件

类层次结构图:

```
java.lang.Object
    --java.awt.Component
        --java.awt.Container
            --javax.swing.JComponent
                --javax.swing.AbstractButton
                    --javax.swing.JToggleButton
                        --javax.swing.JCheckBox
```

我们在上一章节中已经详细地介绍了 JToggleButton 组件,现在我们来介绍 JToggleButton 的两个子类: JCheckBox 与 JRadioButton。JCheckBox 是 JToggleButton 的一个子类,因此它也可以使用 AbstractButton 抽象类里面许多好用的方法,如 addItemListener()、setText()、isSelected()等等。JCheckBox 使用到的地方相当多,如问卷的设计,通常一份问卷里的答案可能是多选,也可能是单选;若多选时便可利用 JCheckBox,而单选时可改用 JRadioButton。JRadioButton 将在下一节介绍。

在使用 JCheckBox 之前,我们先看看 JCheckBox 有哪些构造函数可以使用,如表 7-1 所示。

表 7-1

JCheckBox 的构造函数	
JCheckBox()	建立一个新的 JCheckBox
JCheckBox(Icon icon)	建立一个有图像但没有文字的 JCheckBox
JCheckBox(Icon icon, boolean selected)	建立一个有图像但没有文字的 JCheckBox, 且设置其初始状态 (有无被选取)
JCheckBox(String text)	建立一个有文字的 JCheckBox
JCheckBox(String text, boolean selected)	建立一个有文字的 JCheckBox, 且设置其初始状态 (有无被选取)
JCheckBox(String text, Icon icon)	建立一个有文字且有图像的 JCheckBox, 初始状态为未被选取
JCheckBox(String text, Icon icon, boolean selected)	建立一个有文字且有图像的 JCheckBox, 且设置其初始状态 (有无被选取)

7-1-1 构造 JCheckBox 组件

由上面 JCheckBox 所提供的构造函数可以看出,我们能够很方便地构造出具有图形的 JCheckBox 出来,并设置它的初始状态。以下为一个简单的 JCheckBox 的范例,我们将显示出具有文字与图形的 JCheckBox 组件:

范例 JCheckBox1.java (文件位于随书光盘目录 exam\ch7\JCheckBox1.java)

```
1 import java.awt.*;
2 import java.awt.event.*;
```

复选框、选项按钮、列表方框、下拉式列表的使用与介绍

```
3    import javax.swing.*;
4
5    public class JCheckBox1
6    {
7        public static void main(String args[])
8        {
9            JFrame f = new JFrame("JCheckBox");
10           Container contentPane = f.getContentPane();
11           contentPane.setLayout(new GridLayout(2,1));
12           JPanel p1 = new JPanel();
13           p1.setLayout(new GridLayout(1,3));
14           p1.setBorder(BorderFactory.createTitledBorder("您最喜欢哪一家快餐店呢?"));
15           JCheckBox c1 = new JCheckBox("麦当劳");
16           JCheckBox c2 = new JCheckBox("肯德基");
17           JCheckBox c3 = new JCheckBox("21 世纪");
18           p1.add(c1);
19           p1.add(c2);
20           p1.add(c3);
21           JPanel p2 = new JPanel();
22           p2.setLayout(new GridLayout(2,1));
23           p2.setBorder(BorderFactory.createTitledBorder("以下为 JCheckBox 的图形示范:"));
24           JCheckBox c4 = new JCheckBox("图形 1",new ImageIcon(".\\icons\\-x.jpg"));
25           JCheckBox c5 = new JCheckBox("图形 2",new ImageIcon(".\\icons\\-x.jpg"));
26           p2.add(c4);
27           p2.add(c5);
28           contentPane.add(p1);
29           contentPane.add(p2);
30           f.pack();
31           f.show();
32           f.addWindowListener(new WindowAdapter() {
33               public void windowClosing(WindowEvent e) {
34                   System.exit(0);
35               }
36           });
37       }
38   }
```

◆ 说明:

- (1) 程序第 11 行, 我们将 contentPane 版面设为 2 行 1 列的 GridLayout。
- (2) 程序第 12 行, 我们产生一个 JPanel, 将后面的 3 个 JCheckBox 放进此 JPanel 中, 我们将此 JPanel 设为 1 列 3 行的 GridLayout。
- (3) 程序第 14 行, 我们在此 JPanel 上加入边框 (Border), 并在边框上面加上“您最喜欢哪一家速食店呢?”的文字。
- (4) 程序第 15~20 行, 产生 3 个 JCheckBox, 并依次加入 JPanel 中。
- (5) 程序第 26~29 行, 产生图形选项的 JCheckBox。

✎ 程序运行结果如图 7-1 所示。

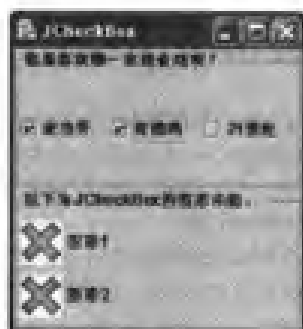


图 7-1

7-1-2 JCheckBox 事件处理

您可以在上面的选项中勾选您喜欢吃的快餐店；在勾选的过程中，您可以发现它是可以复选的。但在图形选项中，我们并无法清楚知道到底用户是否选择了此项目，因为选或不选图形都一样。为解决这个问题，我们要使用到事件处理方法。当 JCheckBox 中的选项被选取或取消时，它会触发 ItemEvent 的事件，ItemEvent 这个类共提供了 4 种方法可以使用，分别是 getItem()、getItemSelectable()、getStateChange()、 paramString()。getItem()与 paramString()方法会返回一些 JCheckBox 的状态值，如下面所示，一般我们很少用到这两种方法：

```
javax.swing.JCheckBox[,5,21,195x41,invalid,layout=javax.swing.OverlayLayout,alignmentX=0.0,alignmentY=0.5,border=javax.swing.plaf.BorderUIResource$CompoundBorderUIResource@5b0afd,flags=1200,maximumSize=,minimumSize=,preferredSize=,defaultIcon=r.jpg,disabledIcon=,disabledSelectedIcon=,margin=javax.swing.plaf.InsetsUIResource[top=2,left=2,bottom=2,right=2],paintBorder=false,paintFocus=true,pressedIcon=,rolloverEnabled=false,rolloverIcon=,rolloverSelectedIcon=,selectedIcon=,text=Java]
```

或是

```
ITEM_STATE_CHANGED,item=javax.swing.JCheckBox[,5,21,195x41,invalid,layout=javax.swing.OverlayLayout,alignmentX=0.0,alignmentY=0.5,border=javax.swing.plaf.BorderUIResource$CompoundBorderUIResource@5b0afd,flags=1200,maximumSize=,minimumSize=,preferredSize=,defaultIcon=r.jpg,disabledIcon=,disabledSelectedIcon=,margin=javax.swing.plaf.InsetsUIResource[top=2,left=2,bottom=2,right=2],paintBorder=false,paintFocus=true,pressedIcon=,rolloverEnabled=false,rolloverIcon=,rolloverSelectedIcon=,selectedIcon=,text=Java],stateChange=SELECTED
```

getItemSelectable()相当于 getSource()方法，一样都是返回触发事件的组件，用来判断是哪个组件产生事件。在上一章中我们曾经说过，getSource()方法是 EventObject 类所提供的，而所有的事件类都会继承这个类，因此所有的事件我们均能用 getSource()方法来判断到底是哪个组件触发了事件。

最后 getStateChange()方法会返回此组件到底有没有被选取。这个方法会返回一个整数值，而我们可以用 ItemEvent 所提供的类变量：若被选取则返回 SELECTED，若没有被选取则返回 DESELECTED。

复选框、选项按钮、列表方框、下拉式列表的使用与介绍

下面这个范例我们即利用 `ItemListener` 来使选取的图形产生变化。`ItemListener` 这个 `Interface` 只定义了一个方法，那就是 `itemStateChanged(ItemEvent e)`，因此我们只需实作这个方法。

范例 JCheckBox2.java (文件位于随书光盘目录 exam\ch7\JCheckBox2.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JCheckBox2 implements ItemListener
6  {
7      JFrame f = null;
8      JCheckBox c4 = null;
9      JCheckBox c5 = null;
10
11     JCheckBox2() {
12         f = new JFrame("JCheckBox");
13         Container contentPane = f.getContentPane();
14         contentPane.setLayout(new GridLayout(2,1));
15         JPanel p1 = new JPanel();
16         p1.setLayout(new GridLayout(1,3));
17         p1.setBorder(BorderFactory.createTitledBorder("您最喜欢哪一家快餐店呢?"));
18         JCheckBox c1 = new JCheckBox("麦当劳");
19         JCheckBox c2 = new JCheckBox("肯德基");
20         JCheckBox c3 = new JCheckBox("21 世纪");
21         p1.add(c1);
22         p1.add(c2);
23         p1.add(c3);
24         JPanel p2 = new JPanel();
25         p2.setLayout(new GridLayout(2,1));
26         p2.setBorder(BorderFactory.createTitledBorder("您喜欢哪种程序语言, 喜欢的请打勾:"));
27         c4 = new JCheckBox("JAVA", new ImageIcon(".\\icons\\x.jpg"));
28         c5 = new JCheckBox("C++", new ImageIcon(".\\icons\\x.jpg"));
29         c4.addItemListener(this);
30         c5.addItemListener(this);
31         p2.add(c4);
32         p2.add(c5);
33         contentPane.add(p1);
34         contentPane.add(p2);
35         f.pack();
36         f.show();
37         f.addWindowListener(new WindowAdapter() {
38             public void windowClosing(WindowEvent e) {
39                 System.exit(0);
40             }
41         });
42     }
43
44     public void itemStateChanged(ItemEvent e)
45     {
46         if (e.getStateChange() == e.SELECTED)
47     {

```

```

48         if(e.getSource() == c4)
49             c4.setIcon(new ImageIcon(".\\icons\\x.jpg"));
50         if(e.getSource() == c5)
51             c5.setIcon(new ImageIcon(".\\icons\\x.jpg"));
52     }
53     }
54     else
55     {
56         if(e.getSource() == c4)
57             c4.setIcon(new ImageIcon(".\\icons\\x.jpg"));
58         if(e.getSource() == c5)
59             c5.setIcon(new ImageIcon(".\\icons\\x.jpg"));
60     }
61 }
62
63 public static void main(String args[])
64 {
65     new JCheckBox2();
66 }
67 }

```

⊕ 说明：

- (1) 程序第 5 行，我们在一开始声明实作 ItemListener Interface，使 JCheckBox2 类具有处理 ItemEvent 的功能。
- (2) 程序第 29、第 30 行，当选取或取消 JCheckBox 时，就会产生 ItemEvent，并交由 ItemListener 来处理。
- (3) 程序第 44~61 行，实作 ItemListener 的 itemStateChanged() 方法。当我们选取选项时，则显示出打勾的图形；若不选取时，则显示出打 X 的图形。

⊕ 程序运行结果如下：

初使状态如图 7-2 所示。

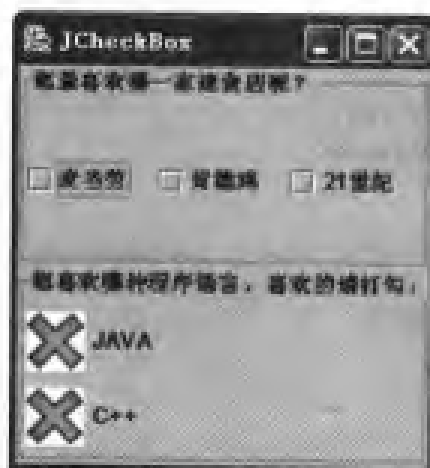


图 7-2

若我们选取了“麦当劳”、“肯德基”与“Java”，则将如图 7-3 所示。

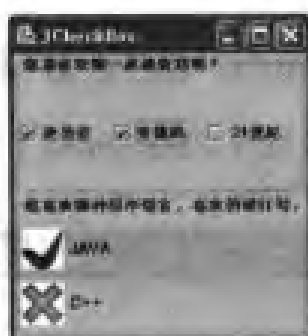


图 7-3

7-2 JRadioButton 的使用

类层次结构图:

```
java.lang.Object
--java.awt.Component
--java.awt.Container
--javax.swing.JComponent
--javax.swing.AbstractButton
--javax.swing.JToggleButton
--javax.swing.JRadioButton
```

JRadioButton 也是 JToggleButton 的一个子类, 因此它也可以使用 AbstractButton 抽象类里面的方法。如同前面所述, JCheckBox 主要用在多重选择的时候, 而 JRadioButton 则是运用在单一选择的时候。例如在设计问卷时, 您想知道测试者的年龄层次, 您就可以用 10~20 岁、21~30 岁、31~40 岁等来区分; 当用户在填写时, 只要选择他在那个年龄层即可。这个问题一定是单选, 一个人不可能又是 20 岁, 又是 40 岁吧!

表 7-2 为 JRadioButton 的构造函数, 您可以发现几乎与 JCheckBox 的构造函数一样, 除了名称不一样之外。

表 7-2

JRadioButton 的构造函数

JRadioButton()	建立一个新的 JRadioButton
JRadioButton (Icon icon)	建立一个有图像但没有文字的 JRadioButton
JRadioButton (Icon icon, boolean selected)	建立一个有图像但没有文字的 JRadioButton, 且设置其初始状态 (有无被选取)
JRadioButton (String text)	建立一个有文字的 JRadioButton
JRadioButton (String text, boolean selected)	建立一个有文字的 JRadioButton, 且设置其初始状态 (有无被选取)
JRadioButton (String text, Icon icon)	建立一个有文字且有图像的 JRadioButton, 初始状态为未被选取
JRadioButton (String text, Icon icon, boolean selected)	建立一个有文字且有图像的 JRadioButton, 且设置其初始状态 (有无被选取)

7-2-1 构造 JRadioButton 组件与事件处理

我们直接来看范例,就知道怎么使用 JRadioButton 了。我们改写上面 JCheckBox 的范例,将 JCheckBox 的部分替换成 JRadioButton:

范例 JRadioButton1.java (文件位于随书光盘目录 exam\ch7\JRadioButton1.java)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JRadioButton1 implements ItemListener
6  {
7      JFrame f = null;
8      JRadioButton r4 = null;
9      JRadioButton r5 = null;
10
11     JRadioButton1() {
12         f = new JFrame("JRadioButton");
13         Container contentPane = f.getContentPane();
14         contentPane.setLayout(new GridLayout(2,1));
15         JPanel p1 = new JPanel();
16         p1.setLayout(new GridLayout(1,3));
17         p1.setBorder(BorderFactory.createTitledBorder("您最喜欢哪一家快餐店呢?"));
18         JRadioButton r1 = new JRadioButton("麦当劳");
19         JRadioButton r2 = new JRadioButton("肯德基");
20         JRadioButton r3 = new JRadioButton("21 世纪");
21         p1.add(r1);
22         p1.add(r2);
23         p1.add(r3);
24         JPanel p2 = new JPanel();
25         p2.setLayout(new GridLayout(2,1));
26         p2.setBorder(BorderFactory.createTitledBorder("您喜欢哪种程序语言,喜欢的请打勾:"));
27         r4 = new JRadioButton("JAVA",new ImageIcon(".\\icons\\x.jpg"));
28         r5 = new JRadioButton("C++",new ImageIcon(".\\icons\\x.jpg"));
29         r4.addItemListener(this);
30         r5.addItemListener(this);
31         p2.add(r4);
32         p2.add(r5);
33         contentPane.add(p1);
34         contentPane.add(p2);
35         f.pack();
36         f.show();
37         f.addWindowListener(new WindowAdapter() {
38             public void windowClosing(WindowEvent e) {
39                 System.exit(0);
40             }
41         });
42     }
43
44     public void itemStateChanged(ItemEvent e)
45     {
```

复选框、选项按钮、列表方框、下拉式列表的使用与介绍

```

46         if (e.getStateChange() == e.SELECTED)
47         {
48             if(e.getSource() == r4)
49                 r4.setIcon(new ImageIcon(".\\icons\\r.jpg"));
50             if(e.getSource() == r5)
51                 r5.setIcon(new ImageIcon(".\\icons\\r.jpg"));
52         }
53         else
54         {
55             if(e.getSource() == r4)
56                 r4.setIcon(new ImageIcon(".\\icons\\x.jpg"));
57             if(e.getSource() == r5)
58                 r5.setIcon(new ImageIcon(".\\icons\\x.jpg"));
59         }
60     }
61 }
62
63 public static void main(String args[])
64 {
65     new JRadioButton1();
66 }
67 }

```

⊕ 说明:

(1) 这个例子我们是改写的上个例子, 将 JCheckBox 替换成 JRadioButton 罢了, 并没有其他的变化。

⊕ 程序运行结果如下:

当我们选取“麦当劳”、“肯德基”、“Java”与“C++”时, 图形显示如图 7-4 所示。



图 7-4

奇怪了……不是说 JRadioButton 是单选吗? 怎么变成多选了呢? 答案揭晓: 其实在 Java 中, JCheckBox 与 JRadioButton 是完全一样的功能, 只是选项的图形不一样罢了 (在不自行设置 Icon 的情况下), 一个是方框的选项, 一个是圆形的选项。Java 提供这两个功能完全一样, 但外观不太一样的组件, 主要是因为用户已经习惯复选就用 JCheckBox, 单选就用 JRadioButton, 这个习惯不管是在软件设计、或是一般 HTML 网页上都可以看到, 所以也就见怪不怪了。因此上面这个范例, 我们应该将 RadioButton 改成单选, 还它原本我们熟悉的特性。

要将 RadioButton 改成单选, 我们必须用到 ButtonGroup 这个类。这个类位于 javax.swing 这个 package 下面, ButtonGroup 类的主要功能是: 同一时间内只会有一个组件的状态为“on”,

其他皆为“off”，也就是同一时间只有一个组件会被选取。而 `ButtonGroup` 类可被 `AbstractButton` 下面的子类所使用，最常被使用的就是 `JRadioButton`、`JRadioButtonMenuItem` 与 `JToggleButton` 这些组件。下面是 `ButtonGroup` 的类层次结构图：

`ButtonGroup` 的类层次结构图：

```
java.lang.Object
    --javax.swing.ButtonGroup
```

我们更改上个范例，使 `RadioButton` 变成单选吧：

范例 JRadioButton2.java (文件位于随书光盘目录 exam\ch7\JRadioButton2.java)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JRadioButton2 implements ItemListener
6  {
7      JFrame f = null;
8      JRadioButton r4 = null;
9      JRadioButton r5 = null;
10
11     JRadioButton2() {
12         f = new JFrame("JRadioButton");
13         Container contentPane = f.getContentPane();
14         contentPane.setLayout(new GridLayout(2,1));
15         JPanel p1 = new JPanel();
16         p1.setLayout(new GridLayout(1,3));
17         p1.setBorder(BorderFactory.createTitledBorder("您最喜欢哪一家快餐店呢?"));
18         JRadioButton r1 = new JRadioButton("麦当劳");
19         JRadioButton r2 = new JRadioButton("肯德基");
20         JRadioButton r3 = new JRadioButton("21 世纪");
21         p1.add(r1);
22         p1.add(r2);
23         p1.add(r3);
24         ButtonGroup bgroupl = new ButtonGroup();
25         bgroupl.add(r1);
26         bgroupl.add(r2);
27         bgroupl.add(r3);
28         JPanel p2 = new JPanel();
29         p2.setLayout(new GridLayout(2,1));
30         p2.setBorder(BorderFactory.createTitledBorder("您喜欢哪种程序语言, 喜欢的请打勾:"));
31         r4 = new JRadioButton("JAVA", new ImageIcon(".\\icons\\x.jpg"));
32         r5 = new JRadioButton("C++", new ImageIcon(".\\icons\\x.jpg"));
33         r4.addItemListener(this);
34         r5.addItemListener(this);
35         p2.add(r4);
36         p2.add(r5);
37         ButtonGroup bgroupp2 = new ButtonGroup();
38         bgroupp2.add(r4);
39         bgroupp2.add(r5);
40         contentPane.add(p1);
41         contentPane.add(p2);
```

```

42         f.pack();
43         f.show();
44         f.addWindowListener(new WindowAdapter() {
45             public void windowClosing(WindowEvent e) {
46                 System.exit(0);
47             }
48         });
49     }
50
51     public void itemStateChanged(ItemEvent e)
52     {
53         if (e.getStateChange() == e.SELECTED)
54         {
55             if(e.getSource() == r4)
56                 r4.setIcon(new ImageIcon(".\\icons\\r.jpg"));
57             if(e.getSource() == r5)
58                 r5.setIcon(new ImageIcon(".\\icons\\r.jpg"));
59         }
60         else
61         {
62             if(e.getSource() == r4)
63                 r4.setIcon(new ImageIcon(".\\icons\\x.jpg"));
64             if(e.getSource() == r5)
65                 r5.setIcon(new ImageIcon(".\\icons\\x.jpg"));
66         }
67     }
68 }
69
70 public static void main(String args[])
71 {
72     new JRadioButton2();
73 }
74 }

```

⊕ 说明:

- (1) 程序第 25 行, 产生一个 ButtonGroup 对象, 并利用 ButtonGroup 类所提供的 add() 方法, 将 3 个 RadioButton 对象放进 ButtonGroup 中, 即程序第 25~27 行的部分。表示此 3 个 RadioButton 同一时间只有一个 RadioButton 的状态可以为“on”。
- (2) 程序第 38~39 行也相同。

⊕ 运行结果如图 7-5 所示。



图 7-5

您可以发现，不管您怎么选，就只能选某一家快餐店或某一种程序语言。

也许您会有这样的疑问，若我不用 `JRadioButton`，而是用 `JCheckBox` 配合 `ButtonGroup`，不是一样可以达到单选的效果吗？没错，您是可以这样做，只是因为大家已经习惯了这样的界面，因此有时候也要考虑一下用户的使用习惯，毕竟约定俗成的东西，不是那么容易改变的嘛！

7-3 JList 的使用

类层次结构图：

```
java.lang.Object
--java.awt.Component
--java.awt.Container
--javax.swing.JComponent
--javax.swing.JList
```

`JList` 与 `JCheckBox` 有点相似，都可以让您选择一到多个选项，较不同的是，`JList` 的选取方式是整列选取。我们先来看看 `JList` 所提供的构造函数，方便迅速建立 `JList` 对象，如表 7-3 所示。

表 7-3

JList 的构造函数	
<code>JList()</code>	建立一个新的 <code>JList</code> 组件
<code>JList(ListModel dataModel)</code>	利用 <code>ListModel</code> 建立一个新的 <code>JList</code> 组件
<code>JList(Object[] listData)</code>	利用 <code>Array</code> 对象建立一个新的 <code>JList</code> 组件
<code>JList(Vector listData)</code>	利用 <code>Vector</code> 对象建立一个新的 <code>JList</code> 组件

7-3-1 建立一般的 JList

一般我们若不需要在 `JList` 中加入 `Icon` 图像，通常会用第 3 或 4 个构造函数来建立 `JList` 对象。而这两者最大的不同在于使用 `Array` 对象建立 `JList` 组件就无法改变项目的数量。对于项目数量经常改变的环境来说，以 `Vector` 对象来建立 `JList` 组件当然比较合适。例如一个卖手机的店家，可能动不动就会有新手机上市，此时若用 `Array` 对象就不是很适当了。

我们来看个范例了解怎么构造一个简单的 `JList` 吧：

范例 `JList1.java` (文件位于随书光盘目录 `exam\ch7\JList1.java`)

```
1 import java.awt.event.*;
2 import javax.swing.*;
3 import java.util.Vector;
4
5 public class JList1
6 {
7     public static void main(String args[])
8     {
```

复选框、选项按钮、列表方框、下拉式列表的使用与介绍

```

9      JFrame f = new JFrame("JList");
10     Container contentPane = f.getContentPane();
11     contentPane.setLayout(new GridLayout(1,2));
12     String[] s = {"美国","日本","中国","英国","法国"};
13     Vector v = new Vector();
14     v.addElement("Nokia 8850");
15     v.addElement("Nokia 8250");
16     v.addElement("Motorola V8088");
17     v.addElement("Motorola V3688x");
18     v.addElement("Panasonic GD92");
19     v.addElement("其他");
20
21     JList list1 = new JList(s);
22     list1.setBorder(BorderFactory.createTitledBorder("您最喜欢到哪个国家玩呢?"));
23
24
25     JList list2 = new JList(v);
26     list2.setBorder(BorderFactory.createTitledBorder("您最喜欢哪一种手机呢?"));
27
28     contentPane.add(list1);
29     contentPane.add(list2);
30     f.pack();
31     f.show();
32     f.addWindowListener(new WindowAdapter() {
33     public void windowClosing(WindowEvent e) {
34         System.exit(0);
35     }
36     });
37 }
38 }

```

⊕ 说明:

- (1) 程序第 12 行, 建立一个 String array, 放置 5 个国家的名称, 当作 JList 的项目值。
- (2) 程序第 13~19 行, 建立一个 Vector, 并在此 Vector 中加入 6 个元素 (Elements), 当作 JList 的项目值。
- (3) 程序第 30~31 行, 将两个 JList 放入 contentPane 中。

⊕ 程序运行结果如图 7-6 所示。

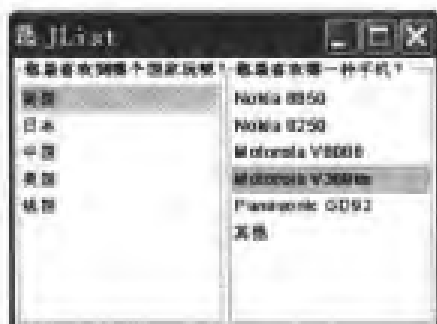


图 7-6

在这个范例中，当窗口变小时，JList 并不会有滚动（ScrollBar）的效果，所以可能无法看到下面的选项。若我们要增加滚动的效果，必须将 JList 放入滚动面板中（JScrollPane），如我们将程序第 28 与第 29 行改成：

```
contentPane.add(new JScrollPane(list1));  
contentPane.add(new JScrollPane(list2));
```

④ 程序运行结果如下：

如此就有滚动的效果了，如图 7-7 所示。

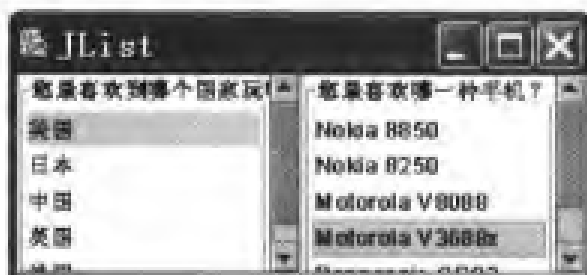


图 7-7

若我们需要有多个选项呢？在 JList 中有 3 种选择模式（Selection Mode）可供我们使用，分别是单一选择、连续区间选择、与多重选择。我们可以在 ListSelectionModel 这个 Interface 中找到这 3 个常数值，如下：

```
static int SINGLE_SELECTION  
static int SINGLE_INTERVAL_SELECTION  
static int MULTIPLE_INTERVAL_SELECTION
```

单一选择：一次只能选一个项目。

连续区间选择：按住【Shift】键，可以对某一连续的项目作选取。

多重选择：没有任何限制。可作单一选择、连续区间选择（按住【Shift】键）、或在不连续的项目作多重选择（按住【Ctrl】键）。多重选择是 Java 对 JList 的默认值，因此在上例中您可以在 JList 中作这 3 种模式的选择方式。

设置选择模式可以利用 JList 所提供的 setSelectionMode(int selectionMode) 方法。例如，若我们将上例改成如下：

范例 JList2.java (文件位于随书光盘目录 exam\ch7\JList2.java)

```
1  import java.awt.*;  
2  import java.awt.event.*;  
3  import javax.swing.*;  
4  import java.util.Vector;  
5  
6  public class JList2  
7  {  
8      public static void main(String args[])  
9      {  
10         JFrame f = new JFrame("JList");  
11         Container contentPane = f.getContentPane();  
12         contentPane.setLayout(new GridLayout(1,3));  
13         String[] s1 = {"美国","日本","中国","英国","法国","意大利","澳洲","韩国"};
```

复选框、选项按钮、列表方框、下拉式列表的使用与介绍

```

14  String[] s2 = {"黄平洋","许铭杰","曹锦辉","曹峻杨","许竹见","郭泓志",
15              "戴龙水","王劲力","其他"};
16  Vector v = new Vector();
17  v.addElement("Nokia 3310");
18  v.addElement("Nokia 8850");
19  v.addElement("Nokia 8250");
20  v.addElement("Motorola V8088");
21  v.addElement("Motorola V3688x");
22  v.addElement("Panasonic GD92");
23  v.addElement("Panasonic GD93");
24  v.addElement("NEC DB2100");
25  v.addElement("Alcatel OT500");
26  v.addElement("Philips Xenium 909 ");
27  v.addElement("Ericsson T29sc");
28  v.addElement("其他");
29
30  JList list1 = new JList(s1);
31  list1.setBorder(BorderFactory.createTitledBorder("您最喜欢到哪个国家玩呢?"));
32
33  JList list2 = new JList(s2);
34  list2.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
35  list2.setBorder(BorderFactory.createTitledBorder("您最喜欢哪个运动员呢?"));
36
37  JList list3 = new JList(v);
38  list3.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
39  list3.setBorder(BorderFactory.createTitledBorder("您最喜欢哪一种手机呢?"));
40
41  contentPane.add(new JScrollPane(list1));
42  contentPane.add(new JScrollPane(list2));
43  contentPane.add(new JScrollPane(list3));
44  f.pack();
45  f.show();
46  f.addWindowListener(new WindowAdapter() {
47      public void windowClosing(WindowEvent e) {
48          System.exit(0);
49      }
50  });
51 }
52 }

```

说明:

- (1) list1 我们未设置何种选择模式, 因此默认值即为 MULTIPLE_INTERVAL_SELECTION。程序第 34 行, 我们设置 list2 为 SINGLE_SELECTION 模式。程序第 38 行, 我们设置 list3 为 SINGLE_INTERVAL_SELECTION 模式。

程序运行结果如下:

由上面的设置可知, list1 可多选、单选或连续区间选取, list2 只能单选, list3 可以单选与连续区间选取, 如图 7-8 所示。



图 7-8

7-3-2 利用 ListModel 构造 JList

ListModel 是一个 Interface，主要的功能是定义一些方法，让 JList 或 JComboBox 这些组件取得每个项目的值，并可限定项目的显示时机与方式。表 7-4 为 ListModel 这个 Interface 所定义的方法：

表 7-4

ListModel Interface 定义的方法	
void	addListDataListener(ListDataListener l) 当 data model 的长度或内容的值有任何改变时，利用此方法就可以处理 ListDataListener 的事件。data model 是 vector 或 array 的数据类型，里面存放 List 中的值
Object	getElementAt(int index) 返回在 index 位置的 item 值
int	getSize() 返回 List 的长度
void	removeListDataListener(ListDataListener l) 删除 ListDataListener

还记得我们一开始在介绍 JList 时所提到的构造函数吗？其中有一个 JList 的构造函数是这样的：

```
JList(ListModel dataModel)
```

因此我们必须实作 ListModel 所有的方法，才能利用上面这个构造函数建立 JList。不过要实现 ListModel 所有的方法有点麻烦，因为一般我们不会用到 addListDataListener() 与 removeListDataListener() 这两个方法。因此 Java 提供了 AbstractListModel 这个抽象类，此抽象类实作了 addListDataListener() 与 removeListDataListener() 这两种方法。若我们继承 AbstractListModel，就不需要实现这两种方法，只需要实作 getElementAt() 与 getSize() 方法即可。我们来看下面的范例：

范例 JList3.java (文件位于随书光盘目录 exam\ch7\JList3.java)

```
1 import java.awt.*;
2 import java.awt.event.*;
```

复选框、选项按钮、列表方框、下拉式列表的使用与介绍

```

3      import javax.swing.*;
4
5
6      public class JList3
7      {
8          public JList3()
9          {
10             JFrame f = new JFrame("JList");
11             Container contentPane = f.getContentPane();
12
13             ListModel mode = new DataModel();
14             JList list = new JList(mode);
15             list.setVisibleRowCount(5);
16             list.setBorder(BorderFactory.createTitledBorder("您最喜欢到哪个国家玩呢?"));
17
18             contentPane.add(new JScrollPane(list));
19             f.pack();
20             f.show();
21             f.addWindowListener(new WindowAdapter() {
22                 public void windowClosing(WindowEvent e) {
23                     System.exit(0);
24                 }
25             });
26         }
27
28         public static void main(String args[])
29         {
30             new JList3();
31         }
32     }
33
34     class DataModel extends AbstractListModel
35     {
36         String[] s = {"美国", "日本", "中国", "英国", "法国", "意大利", "澳洲", "韩国"};
37         public Object getElementAt(int index)
38         {
39             return (index+1)+ "." + s[index++];
40         }
41
42         public int getSize()
43         {
44             return s.length;
45         }
46     }

```

⊕ 说明:

- (1) 程序第 13 行, 由于我们在 DataModel 类中继承 AbstractListModel, 并实作了 getElementAt() 与 getSize() 方法, 因此我们可以由 DataModel 类产生一个 ListModel 的实体来。
- (2) 程序第 14 行, 利用 ListModel 建立一个 JList。
- (3) 程序第 15 行, 设置程序一打开时所能看到的数据项个数。

- (4) 由于我们在程序第 34 行继承 `AbstractListModel` 抽象类, 因此我们分别在程序第 37~40 行与程序第 42~45 行实作了 `getElementAt()` 与 `getSize()` 方法。
- (5) 当程序在第 20 行要 `show` 出 `list` 时, 系统会先自动调用 `getSize()` 方法, 看看这个 `list` 长度有多少; 然后再调用 `setVisibleRowCount()` 方法, 看要一次输出几笔数据; 最后调用 `getElementAt()` 方法, 将 `list` 中的项目值 (`item`) 填入 `list` 中。读者若还是不太清楚, 可直接在 `getSize()` 与 `getElementAt()` 方法中个别加入 `System.out.println("size")` 与 `System.out.println("element")` 叙述, 就可以在 `dos console` 中清楚看出整个显示 `list` 调用的过程。
- (6) `getElementAt()` 方法中的参数 `index`, 系统会自动由 0 开始计算, 不过要自己作累加的操作, 如程序第 39 行所示。

④ 程序运行结果如图 7-9 所示。



图 7-9

事实上, Java 本身还提供了另一个类, `DefaultListModel` 实体类。此类继承了 `AbstractListModel` 抽象类, 并实现里面所有的抽象方法, 因此您不需要再自行实作任何的方法, 可说是相当的方便。不过既然所有的抽象方法都被实现, 因此在设计的弹性上就会有所降低。若您喜欢自行管理 `JList` 项目的设计者, 您可以不要使用 `DefaultListModel` 这个类, 只需要在 `AbstractListModel` 上多下功夫即可。

另外, `DefaultListModel` 类具有 `Vector` 的相关功能, 因此若您要加入任何项目, 您可以用 `addElement()` 方法, 将所要的项目加入 `JList` 中。

下面这个例子改写自上个例子, 直接使用 `DefaultListModel` 类:

范例 `JList4.java` (文件位于随书光盘目录 `exam\ch7\JList4.java`)

```
1  import java.awt.*;  
2  import java.awt.event.*;  
3  import javax.swing.*;  
4  
5  
6  public class JList4  
7  {  
8      public JList4()  
9      {  
10         JFrame f = new JFrame("JList");
```

复选框、选项按钮、列表方框、下拉式列表的使用与介绍

本章主要介绍复选框、选项按钮、列表方框、下拉式列表的使用与介绍。本章内容分为两部分：第一部分介绍复选框、选项按钮、列表方框、下拉式列表的使用；第二部分介绍复选框、选项按钮、列表方框、下拉式列表的编程。

```

11         Container contentPane = f.getContentPane();
12
13         ListModel mode = new DataModel();
14         JList list = new JList(mode);
15         list.setVisibleRowCount(5);
16         list.setBorder(BorderFactory.createTitledBorder("您最喜欢到哪个国家玩呢?"));
17
18         contentPane.add(new JScrollPane(list));
19         f.pack();
20         f.show();
21         f.addWindowListener(new WindowAdapter() {
22             public void windowClosing(WindowEvent e) {
23                 System.exit(0);
24             }
25         });
26     }
27
28     public static void main(String args[])
29     {
30         new JList4();
31     }
32 }
33
34 class DataModel extends DefaultListModel
35 {
36     String[] s = {"美国", "日本", "中国", "英国", "法国", "意大利", "澳洲", "韩国"};
37
38     DataModel()
39     {
40         for(int i=0; i < s.length; i++)
41             addElement((i+1)+ "." + s[i]);
42     }
43 }

```

说明：

- (1) 程序第 34 行，我们使 DataModel 继承 DefaultListModel 实体类，因此就不需要再实作 getSize() 与 getElementAt() 这两种方法，直接将所要的项目用 addElement() 方法加入即可。系统会自动将所加入的项目放进一个 Vector 对象中，并在输出 JList 时自动调用 getSize() 与 getElementAt() 方法。
- (2) 您可以在 DefaultListModel 类中找到 getSize() 与 getElementAt() 这两种方法，您可以发现这两种方法已经被实作，不再是抽象方法了。

程序运行结果与上个范例相同。

很奇怪，这不是跟我们使用 Vector 方式，利用 JList(Vector v) 构造函数来建立新的 JList 一样吗？如同 JList1.java 中的例子，为什么还要多此一举呢？其实若读者去察看 DefaultListModel 类，可发现此类提供不少好用的方法，例如您可以随意的增加一个项目 (addElement())、或是删除一个项目 (removeElement())、甚至可以很方便地做到查询 (getElementAt()) 与汇出 (copyInto()) 项目的操作。您会发现，利用 DefaultListModel 可以直接动

态地更改 JList 的项目值，而不需要自行产生一个 Vector 对象；相对于 JList(Vector v) 这个构造函数，可说更方便且实用许多。

至于利用 ListModel 或 AbstractListModel 来构造 JList 有什么好处？读者只要这样想，ListModel 中文就是“列表模式”，那么套用不同的“列表模式”，就会有不同的显示效果啰？没错，一点也没错，例如每个老师都会有自己所开课目的学生成绩，老师可以看到每个同学的成绩，而学生只能看到自己的成绩，所以我们就会有两种不同的“列表模式”。我们只需要去改写 getElementAt() 方法，就会有不同的“列表模式”产生，如下面这个例子：

范例 JList5.java (文件位于随书光盘目录 exam\ch7\JList5.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5
6  public class JList5
7  {
8      public JList5()
9      {
10         JFrame f = new JFrame("JList");
11         Container contentPane = f.getContentPane();
12         contentPane.setLayout(new GridLayout(1,2));
13         ListModel mode = new DataModel(1);
14         JList list1 = new JList(mode);
15         list1.setVisibleRowCount(5);
16         list1.setBorder(BorderFactory.createTitledBorder("您玩过哪些软件?"));
17
18         mode = new DataModel(2);
19         JList list2 = new JList(mode);
20         list2.setBorder(BorderFactory.createTitledBorder("您玩过哪些数据库软件?"));
21
22         contentPane.add(new JScrollPane(list1));
23         contentPane.add(new JScrollPane(list2));
24         f.pack();
25         f.show();
26         f.addWindowListener(new WindowAdapter() {
27             public void windowClosing(WindowEvent e) {
28                 System.exit(0);
29             }
30         });
31     }
32
33     public static void main(String args[])
34     {
35         new JList5();
36     }
37 }
38
39 class DataModel extends AbstractListModel
40 {
41     String[] s = {"MS SQL","MySQL","IBM DB2","ORACLE","Windows
42                 2000","Linux","UNIX","OS2"};

```

```
43     int flag;  
44  
45     DataModel(int flag)  
46     {  
47         this.flag = flag;  
48     }  
49     public Object getElementAt(int index)  
50     {  
51         String tmp = null;  
52  
53         if (flag == 1)  
54             tmp = (index+1)+","+s[index++];  
55         if (flag == 2)  
56         {  
57             if(index < 4)  
58                 tmp = (index+1)+","+s[index++];  
59         }  
60  
61         return tmp;  
62     }  
63  
64     public int getSize()  
65     {  
66         return s.length;  
67     }  
68 }
```

⊕ 说明:

- (1) 我们分别在程序第 13 行与第 19 行产生两种不同的 ListModel, 并将这两种不同的 ListModel 放入不同的 JList 中。
- (2) 程序第 49~62 行, 改写 getElementAt()方法, 当 JList 的标题是“您玩过哪些软件”时, 则列出 String array s 的所有数据, 当 JList 的标题是“您玩过哪些数据库软件”时, 则只列出 String array s 中的前 4 项数据。

⊕ 程序运行结果如图 7-10 所示。



图 7-10

7-3-3 建立有图像的 JList

我们也可以在 JList 中加入 Icon 图像,不过在 JList 中加入图像比较麻烦一点,不像 JLabel 或 JButton 那样简单。要在 JList 上加入 Icon,要先了解 ListCellRenderer Interface。我们必须由这个 Interface 所定义的方法,将图像画在 JList 中的每个项目。ListCellRenderer Interface 里只定义了一种方法,那就是 getListCellRendererComponent,不过这种方法参数有点多,我们把它列出来看一下:

```
public Component getListCellRendererComponent (JList list,  
                                                Object value,  
                                                int index,  
                                                boolean isSelected,  
                                                boolean cellHasFocus)
```

list: 即所要画上图像的 JList 组件。

value: JList 项目的值,如 list.getModel().getElementAt(index)所返回的值。

index: 为 JList 项目的索引值,由 0 开始。

isSelected 与 cellHasFocus: 判断 JList 中的项目是否有被选取或是有焦点置入。

上面这 4 个参数会在您设置 JList 的绘图样式 (setCellRenderer()) 时自动的由 JList 组件提供,您只需关心怎么控制 getListCellRendererComponent()方法中的 4 个参数,而无需担心怎么传入参数。

要在 JList 中加入 Icon 图像的技巧就是将 JList 中的每一个项目当作是 JLabel,因为 JLabel 在使用文字与图像上非常的方便。要设置 JList 的图像,必须使用 setCellRenderer(ListCellRenderer cellRenderer)这个方法。我们来看下面这个范例,您就能明白了:

范例 JList6.java (文件位于随书光盘目录 exam\ch7\JList7.java)

```
1  import java.awt.*;  
2  import java.awt.event.*;  
3  import javax.swing.*;  
4  
5  
6  public class JList6  
7  {  
8      public JList6()  
9      {  
10         String[] s = {"西瓜","苹果","草莓","香蕉","葡萄"};  
11         JFrame f = new JFrame("JList");  
12         Container contentPane = f.getContentPane();  
13         JList list1 = new JList(s);  
14         list1.setBorder(BorderFactory.createTitledBorder("您喜欢吃哪些水果?"));  
15         list1.setCellRenderer(new CellRenderer());  
16  
17         contentPane.add(new JScrollPane(list1));  
18         f.pack();  
19         f.show();  
20         f.addWindowListener(new WindowAdapter() {
```

复选框、选项按钮、列表方框、下拉式列表的使用与介绍

```

21         public void windowClosing(WindowEvent e) {
22             System.exit(0);
23         }
24     });
25 }
26
27     public static void main(String args[])
28     {
29         new JList6();
30     }
31 }
32
33     class CellRenderer extends JLabel implements ListCellRenderer
34     {
35         CellRenderer()
36         {
37             setOpaque(true);
38         }
39
40         public Component getListCellRendererComponent(JList list,
41                                                         Object value,
42                                                         int index,
43                                                         boolean isSelected,
44                                                         boolean cellHasFocus)
45         {
46             if (value != null)
47             {
48                 setText(value.toString());
49                 setIcon(new ImageIcon(".\\icons\\fruit" + (index + 1) + ".jpg"));
50             }
51             if (isSelected) {
52                 setBackground(list.getSelectionBackground());
53                 setForeground(list.getSelectionForeground());
54             }
55             else {
56                 setBackground(list.getBackground());
57                 setForeground(list.getForeground());
58             }
59
60             return this;
61         }
62     }

```

⊕ 说明:

- (1) 程序第 15 行, 设置在 JList 中画上图像。在此参数中, 我们产生一个 CellRenderer 对象, 此对象实作了 ListCellRenderer Interface, 因此可以返回一个 ListCellRenderer 对象作为 setCellRenderer() 方法的参数。
- (2) 程序第 33 行, 类 CellRenderer 继承 JLabel 并实作 ListCellRenderer。由于我们利用 JLabel 易于插图的特性, 因此 CellRenderer 继承了 JLabel, 可让 JList 中的每个项目都被视为是一个 JLabel。
- (3) 程序第 40~61 行, 实作 getListCellRendererComponent() 方法。

- (4) 在第 46 行中，我们判断 `list.getModel().getElementAt(index)` 所返回的值是否为 `null`，例如在上个例子中，若 `JList` 的标题是“您玩过哪些数据库软件”，则 `index >= 4` 的项目值我们全都设为 `null`。而在这个例子中，因为不会有 `null` 值，所以有没有加上这个判断并没有关系。
- (5) 程序第 49 行，我们所使用的图文件依次是 `fruit1.jpg~fruit4.jpg`。
- (6) 程序第 51~58 行，设置选取与取消选取的前景与背景颜色。

✎ 程序运行结果如图 7-11 所示。



图 7-11

您必须使用 `setOpaque(true)` 方法才能在选择项目时显示出反白效果。若您将程序第 37 行 `Mark` 起来或设为 `false`，再重新编译与运行，则您会发现怎么选都不会有紫蓝色选取长条出现，因为默认值为透明色是看不见的，如图 7-12 所示。



图 7-12

除此之外，您也要记得设置前景与背景颜色。若您将程序第 51~58 行 `mark` 起来，重新编译与运行，您会发现前后景都不见了，且再怎么选取颜色也不会有变化（见图 7-13）。因此在编写这类程序时，可不要忘了这个小细节喔。

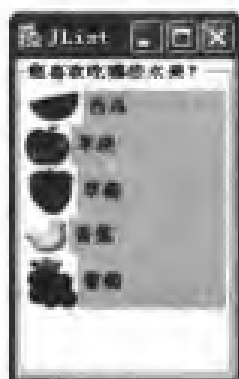


图 7-13

7-3-4 JList 的事件处理

JList 的事件处理一般可分为两种：一种是取得用户选取的项目；另一种是在 JList 的项目上双击鼠标两次，运行相对应的工作。我们先来看第一种事件处理方式。

在 JList 类中有 addListSelectionListener() 方法，可以检测用户是否对 JList 的选取有任何的改变。ListSelectionListener Interface 中只定义了一种方法，那就是 valueChanged (ListSelectionEvent e)，我们必须实作这个方法，才能在用户改变选取值时取得用户最后的选取状态。我们来看下面这个例子：

这个范例使我们能抓取用户所选取的项目，并将选取的项目显示在 JLabel 上。

范例 JList7.java (文件位于随书光盘目录 exam\ch7\ JList7.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import javax.swing.event.*;
5
6  public class JList7 implements ListSelectionListener
7  {
8      JList list = null;
9      JLabel label = null;
10     String[] s = {"美国", "日本", "中国", "英国", "法国", "意大利", "澳洲", "韩国"};
11
12     public JList7()
13     {
14         JFrame f = new JFrame("JList");
15         Container contentPane = f.getContentPane();
16         contentPane.setLayout(new BorderLayout());
17         label = new JLabel();
18
19         list = new JList(s);
20         list.setVisibleRowCount(5);
21         list.setBorder(BorderFactory.createTitledBorder("您最喜欢到哪个国家玩呢?"));
22         list.addListSelectionListener(this);
23
24         contentPane.add(label, BorderLayout.NORTH);
25         contentPane.add(new JScrollPane(list), BorderLayout.CENTER);
26         f.pack();

```

```
27         f.show();
28         f.addWindowListener(new WindowAdapter() {
29             public void windowClosing(WindowEvent e) {
30                 System.exit(0);
31             }
32         });
33     }
34
35     public static void main(String args[])
36     {
37         new JList7();
38     }
39
40     public void valueChanged(ListSelectionEvent e)
41     {
42         int tmp = 0;
43         String stmp = "您目前选取: ";
44         int[] index = list.getSelectedIndices();
45         for(int i=0; i < index.length ; i++)
46         {
47             tmp = index[i];
48             stmp = stmp+s[tmp]+" ";
49         }
50         label.setText(stmp);
51     }
52 }
```

◆ 说明:

- (1) 程序第 4 行, 由于 ListSelectionEvent 是 swing 的事件, 不是 awt 的事件, 因此我们必须先 import javax.swing.event.*, 否则系统会不认得什么是 ListSelectionEvent。
- (2) 程序第 6 行, 我们在此类中实作 ListSelectionListener 这个 Interface。
- (3) 程序第 22 行, addListSelectionListener() 使这个 JList 具有检测用户改变选项的功用。当用户改变选取值时, 系统便会调用 valueChanged() 方法来处理选项的改变。
- (4) 程序第 40~51 行, 实作 valueChanged() 方法。
- (5) 程序第 44 行, 利用 JList 类所提供的 getSelectedIndices() 方法, 可得到用户所选取的所有 index 值, 这些 index 值由一个 int array 返回。

◆ 程序运行结果如图 7-14 所示。



图 7-14

复选框、选项按钮、列表方框、下拉式列表的使用与介绍

接下来,我们来看如何处理在 JList 上双击鼠标的操作。由于 JList 本身并无提供这样的 `EventListener`, 因此我们必须利用 `MouseListener` 来达到捕获双击鼠标事件的目的。至于要怎么知道我们到底在哪个项目上双击鼠标呢? 我们可以利用 JList 类所提供的 `locatToindex()` 方法得知。以下为我们所举的范例:

这个例子一开始在左边列有国家名称, 当您在某个国家名称上双击鼠标后, 这个国家名称就会移到右边去, 反之亦然。

范例 JList8.java (文件位于随书光盘目录 exam\ch7\JList8.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import javax.swing.event.*;
5
6  public class JList8 extends MouseAdapter
7  {
8      JList list1 = null;
9      JList list2 = null;
10     DefaultListModel model = null;
11     DefaultListModel mode2 = null;
12     String[] s = {"美国", "日本", "中国", "英国", "法国", "意大利", "澳洲", "韩国"};
13
14     public JList8()
15     {
16         JFrame f = new JFrame("JList");
17         Container contentPane = f.getContentPane();
18         contentPane.setLayout(new GridLayout(1,2));
19
20         model = new DataModel(1);
21         list1 = new JList(model);
22         list1.setBorder(BorderFactory.createTitledBorder("国家名称! "));
23         list1.addMouseListener(this);
24
25         mode2 = new DataModel(2);
26         list2 = new JList(mode2);
27         list2.setBorder(BorderFactory.createTitledBorder("您最喜欢到哪个国家玩呢? "));
28         list2.addMouseListener(this);
29
30         contentPane.add(new JScrollPane(list1));
31         contentPane.add(new JScrollPane(list2));
32         f.pack();
33         f.show();
34         f.addWindowListener(new WindowAdapter() {
35             public void windowClosing(WindowEvent e) {
36                 System.exit(0);
37             }
38         });
39     }
40
41     public static void main(String args[])
42     {
43         new JList8();

```



```

44     }
45
46     public void mouseClicked(MouseEvent e)
47     {
48         int index;
49
50         if (e.getSource() == list1)
51         {
52             if(e.getClickCount() == 2)
53             {
54                 index = list1.locationToIndex(e.getPoint());
55                 String tmp = (String)model.getElementAt(index);
56                 mode2.addElement(tmp);
57                 list2.setModel(mode2);
58                 model.removeElementAt(index);
59                 list1.setModel(model);
60             }
61         }
62         if (e.getSource() == list2)
63         {
64             if(e.getClickCount() == 2)
65             {
66                 index = list2.locationToIndex(e.getPoint());
67                 String tmp = (String)mode2.getElementAt(index);
68                 model.addElement(tmp);
69                 list1.setModel(model);
70                 mode2.removeElementAt(index);
71                 list2.setModel(mode2);
72             }
73         }
74     }
75
76     class DataModel extends DefaultListModel
77     {
78         DataModel(int flag)
79         {
80             if (flag == 1)
81             {
82                 for(int i=0; i < s.length; i++)
83                     addElement(s[i]);
84             }
85         }
86     }
87 }

```

说明:

- (1) 这个范例我们应用了 DefaultListModel 类, 因为 DefaultListModel 类实作了 Vector 中的方法, 使我们处理动态的 JList 项目值比较容易。
- (2) 由于我们要处理鼠标事件, 为了编写方便, 在程序第 6 行我们继承 MouseAdapter 抽象类。
- (3) 程序第 20 行与第 25 行, 建立两个 DataModel, 此类我们写在 76~86 行之间。第一个 JList, 也就是 list1 一开始时会把 String array s 中的所有值依次放入 list1 的项目

复选框、选项按钮、列表方框、下拉式列表的使用与介绍

中，而 list2 一开始为空白。

(4) 程序第 23 与第 28 行，使 JList 遇到鼠标事件时，立即进行处理。

(5) 程序第 46~74，处理鼠标键击事件。

(6) 程序第 50~61 行，对 list1 而言，当鼠标在某个项目上连续按两下时，我们利用 JList 所提供的 locationToIndex() 方法，找到所键击的项目，并在第 55 行取得此项目的项目值，然后将此项目值增加到 mode2 中（第 56 行），重新设置 list2 的 ListModel（第 57 行），使 list2 可显示出所增加的项目。并在第 58~59 行中，将刚刚在 list1 双击的项目删除。

(7) 程序第 62~73 行，作用同上，只是顺序相反。

◆ 程序运行结果如图 7-15 所示。

程序运行后的初始状态：



图 7-15

当我们对“英国”与“法国”项目双击时，即可得到下面结果，如图 7-16 所示。



图 7-16

反之，您也可以在右边的 JList 中双击（Double Click）项目，同样可以把右边的项目移到左边去。

7-4 JComboBox 的使用

类层次结构图：

```
java.lang.Object
--java.awt.Component
--java.awt.Container
```

```
--javax.swing.JComponent
--javax.swing.JComboBox
```

JComboBox 一般称为下拉式列表，可让您在一系列的选项中选出您要的值，或直接输入您要的值。如在 MS Word 中，您可以选择字形的大小或自行输入字体的大小，如图 7-17 所示。



图 7-17

JComboBox 的构造方式跟 JList 很像，甚至连使用方式都有点相像，下面我们会慢慢为您介绍。我们先来看看 JComboBox 所提供的构造函数吧，如表 7-5 所示。

表 7-5

JComboBox 的构造函数	
JComboBox()	建立一个新的 JComboBox 组件
JComboBox(ComboBoxModel aModel)	利用 ListModel 建立一个新的 JComboBox 组件
JComboBox(Object[] items)	利用 Array 对象建立一个新的 JComboBox 组件
JComboBox(Vector items)	利用 Vector 对象建立一个新的 JComboBox 组件

读者可翻回前面所介绍的 JList 章节，是否发现这两个构造方式真的很像呢！

不过在第二个构造函数中，JList 是用 ListModel，而 JComboBox 是用 ComboBoxModel，这个差异我们将在下面介绍。

7-4-1 建立一般的 JComboBox

若我们不需要在 JComboBox 中加入 Icon 图像，如同 JList 一般，我们可以用第 3 或第 4 个构造函数来建立 JComboBox 对象。不过 JComboBox 可利用本身类所提供的 addItem() 方法，为 JComboBox 组件增加选项。我们来看下面的范例：

范例 JComboBox1.java (文件位于随书光盘目录 exam\ch7\ JComboBox1.java)

```
1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
```

复选框、选项按钮、列表方框、下拉式列表的使用与介绍

```

4    import java.util.Vector;
5
6    public class JComboBox1
7    {
8        public static void main(String args[])
9        {
10            JFrame f = new JFrame("JComboBox");
11            Container contentPane = f.getContentPane();
12            contentPane.setLayout(new GridLayout(1,2));
13            String[] s = {"美国","日本","中国","英国","法国"};
14            Vector v = new Vector();
15            v.addElement("Nokia 8850");
16            v.addElement("Nokia 8250");
17            v.addElement("Motorola V8088");
18            v.addElement("Motorola V3688x");
19            v.addElement("Panasonic GD92");
20            v.addElement("其他");
21
22            JComboBox combol = new JComboBox(s);
23            combol.addItem("澳洲");
24            combol.setBorder(BorderFactory.createTitledBorder("您最喜欢到哪个国家玩呢?"));
25
26            JComboBox combo2 = new JComboBox(v);
27            combo2.setBorder(BorderFactory.createTitledBorder("您最喜欢哪一种手机呢?"));
28
29            contentPane.add(combol);
30            contentPane.add(combo2);
31            f.pack();
32            f.show();
33            f.addWindowListener(new WindowAdapter() {
34                public void windowClosing(WindowEvent e) {
35                    System.exit(0);
36                }
37            });
38        }
39    }

```

⊕ 说明:

- (1) 程序第 22 行, 建立一个 JComboBox 组件, 里面的项目由程序第 13 行中的 String array s 所提供。
- (2) 程序第 23 行, 利用 JComboBox 类所提供的 addItem() 方法, 加入一个项目到此 JComboBox 中。
- (3) 程序第 27 行, 建立一个 JComboBox 组件, 里面的项目由程序第 14 行中的 Vector v 所提供。
- (4) 程序第 29~30 行, 将两个 JComboBox 放入 contentPane 中。

④ 程序运行结果如图 7-18 所示。



图 7-18

7-4-2 利用 ComboBoxModel 构造 JComboBox

如同 JList 一般，在 JComboBox 中也有一个构造函数是利用某种 Model 来构造。如下所示：

```
JComboBox(ComboBoxModel aModel)
```

ComboBoxModel 是一个 Interface，里面定义了两种方法，分别是 setSelectedItem() 与 getSelectedItem()。这两种方法目的是让用户选取某个项目后，可以正确地显示出用户所选取的项目。表 7-6 为这两种方法详细地定义。

表 7-6

返回类型	ComboBoxModel Interface 定义的方法
Object	getSelectedItem() 返回所选取的项目值
Void	setSelectedItem(Object anItem) 设置所选取的项目值

与 JList 不同的是，JComboBox 是利用 ComboBoxModel，而不是 ListModel。不过 ComboBoxModel Interface 是继承 ListModel Interface，因此若我们要利用 ComboBoxModel 来构造 JComboBox，除了要实作 ComboBoxModel 的两种方法外，还必须实作 ListModel 所定义的 4 种方法，这样的做法可说是相当麻烦。

在介绍 JList 时我们曾经提到 AbstractListModel 是个抽象类，这个抽象类实现了 ListModel Interface 中的 addListDataListener() 与 removeListDataListener() 这两种方法。因此若我们继承 AbstractListModel，则可不用实现这两种方法，只需要实现 getElementAt()、getSize()、setSelectedItem() 与 getSelectedItem() 这种种方法。这样的做法就显得比较好一点，图 7-19 为这些 Interface 与类间的关系图，由这个图标，读者应该可以很清楚看出它们之间的关系。

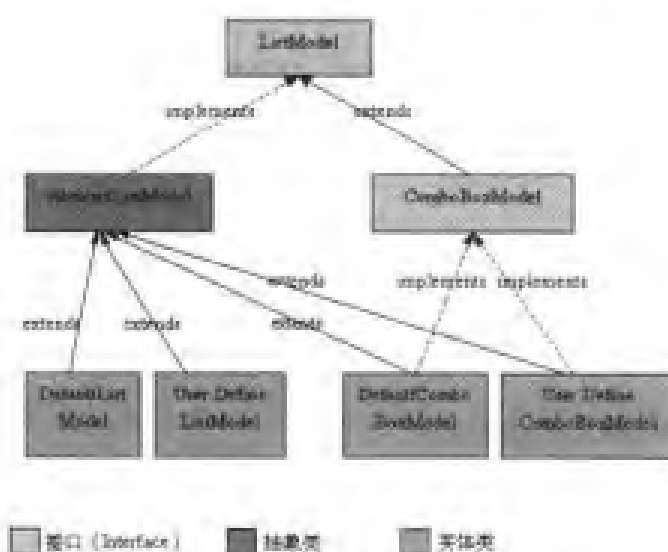


图 7-19

让我们来看看如何利用 ComboBoxModel 来构造 JComboBox:

范例 JComboBox2.java (文件位于随书光盘目录 exam\ch7\ JComboBox2.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JComboBox2
6  {
7      String[] s = {"美国", "日本", "中国", "英国", "法国", "意大利", "澳洲", "韩国"};
8      public JComboBox2()
9      {
10         JFrame f = new JFrame("JComboBox");
11         Container contentPane = f.getContentPane();
12
13         ComboBoxModel mode = new UserDefineComboBoxModel();
14         JComboBox combo = new JComboBox(mode);
15         combo.setBorder(BorderFactory.createTitledBorder("您最喜欢到哪个国家玩呢?"));
16         contentPane.add(combo);
17         f.pack();
18         f.show();
19         f.addWindowListener(new WindowAdapter() {
20             public void windowClosing(WindowEvent e) {
21                 System.exit(0);
22             }
23         });
24     }
25
26     public static void main(String args[])
27     {
28         new JComboBox2();
29     }

```

```
30
31     class UserDefineComboBoxModel extends AbstractListModel implements
    ComboBoxModel
32     {
33         String item = null;
34
35         public Object getElementAt(int index)
36         {
37             return s[index++];
38         }
39
40         public int getSize()
41         {
42             return s.length;
43         }
44
45         public void setSelectedItem(Object anItem)
46         {
47             item = (String)anItem;
48         }
49
50         public Object getSelectedItem()
51         {
52             return item;
53         }
54     }
55 }
```

◆ 说明:

- (1) 程序第 15 行, 由于我们在 UserDefineComboBoxModel 类中继承 AbstractListModel 与实作 ComboBoxModel 界面。我们共实作了 getElementAt()、getSize()、setSelectedItem() 与 getSelectedItem() 这 4 种方法。因此我们可以由 UserDefineComboBoxModel 类产生一个 ComboBoxModel 的实体。
- (2) 程序第 16 行, 利用 ComboBoxModel 建立一个 JComboBox。
- (3) 由于我们在程序第 31 行继承 AbstractListModel 抽象类。因此我们分别在程序第 35~38 行与程序第 40~43 行实作了 getElementAt()与 getSize()方法。
- (4) 由于我们在程序第 31 行 implements ComboBoxModel Interface。因此我们必须在程序第 45~48 行与程序第 50~53 行实作 setSelectedItem()与 getSelectedItem()方法。
- (5) 当程序要 show 出 JComboBox 时, 系统会先自动调用 getSize()方法, 看看这个 JComboBox 的长度有多少, 然后再调用 getElementAt()方法, 将 String Array s 中的值填入 JComboBox 中。当用户选择项目时, 系统会调用 getSelectedItem()方法, 返回所选取的项目, 并利用 setSelectedItem()方法, 将所选取的项目放在 JComboBox 的最前端。
- (6) getElementAt()方法中的 “index” 参数, 系统会自动由 0 开始计算, 不过要自己作累加的操作, 如程序第 37 行所示。

◆ 程序运行结果如图 7-20 所示。



图 7-20

如同 JList 一般, Java 对于 JComboBox 也提供了另一个类, DefaultComboBoxModel 实体类。此类继承了 AbstractListModel 抽象类, 也实现了 ComboBoxModel Interface。因此您不需要再实现 getSize()、getElementAt()、setSelectedItem() 与 getSelectedItem() 方法。利用 DefaultComboBoxModel 这个类我们可以很方便地做到动态更改 JComboBox 的项目值。当您没有必要自己定义特殊的 ComboBoxModel 时, 使用 DefaultComboBoxModel 就显得非常的方便。我们来看下面的例子:

范例 JComboBox3.java (文件位于随书光盘目录 exam\ch7\JComboBox3.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JComboBox3
6  {
7      String[] s = {"美国", "日本", "大陆", "英国", "法国", "意大利", "澳洲", "韩国"};
8
9      public JComboBox3()
10     {
11         JFrame f = new JFrame("JComboBox");
12         Container contentPane = f.getContentPane();
13
14         ComboBoxModel mode = new AModel();
15         JComboBox combo = new JComboBox(mode);
16         combo.setBorder(BorderFactory.createTitledBorder("您最喜欢到哪个国家玩呢?"));
17
18         contentPane.add(combo);
19         f.pack();
20         f.show();
21         f.addWindowListener(new WindowAdapter() {
22             public void windowClosing(WindowEvent e) {

```



```

23         System.exit(0);
24     }
25     });
26 }
27
28 public static void main(String args[])
29 {
30     new JComboBox3();
31 }
32
33 class AModel extends DefaultComboBoxModel
34 {
35     AModel()
36     {
37         for(int i=0; i < s.length; i++)
38             addElement(s[i]);
39     }
40 }
41 }

```

⊕ 说明:

- (1) 程序第 14 行, 由于 AModel 继承 DefaultComboBoxModel 实体类, 由 AModel 可得到一个 ComboBoxModel 实体对象。
- (2) 程序第 33 行, 我们使 AModel 继承 DefaultComboBoxModel 实体类, 因此就不需要再实作 getElementAt()、getSize()、setSelectedItem()与 setSelectedItem()这 4 种方法, 直接将所要的项目用 addElement()方法加入即可。系统会自动将所加入的项目放进一个 Vector 中, 并在输出 JComboBox 时自动调用 getSize()与 getElementAt()方法。程序运行结果与上个范例相同。

7-4-3 建立有图像的 JComboBox

在上一节中, 我们利用 ListCellRenderer Interface 在 JList 中加入 Icon 图像, 而要在 JComboBox 中加入图像的方法也是一样的。我们必须实作 ListCellRenderer Interface 所定义的方法 getListCellRendererComponent。以下为这个方法的定义:

```

public Component getListCellRendererComponent (JList list,
                                                Object value,
                                                int index,
                                                boolean isSelected,
                                                boolean cellHasFocus)

```

参数 list 即所要画上图像的 JComboBox 组件。

参数 value 为 JComboBox.getModel().getElementAt(index)所返回的值。

参数 index 为 JComboBox 中项目的索引值, 由 0 开始。

参数 isSelected 与 cellHasFocus 则是判断 JComboBox 中的项目是否有被选取或是否有焦点置入。

上面这 4 个参数会在您设置 JComboBox 的绘图样式 (setRenderer ()) 时自动的由

复选框、选项按钮、列表方框、下拉式列表的使用与介绍

JComboBox 组件提供, 您只需关心怎么控制 `getListCellRendererComponent()` 方法中的 4 个参数, 而无须担心怎么传入参数。

如同 JList, 要在 JComboBox 中加入 Icon 图像就是将 JComboBox 中的每一个项目当作是一个 JLabel, 因为 JLabel 在使用文字与图像上非常的方便。而要在 JComboBox 上画上图像, 必须使用 `setRenderer(ListCellRenderer cellRenderer)` 这个方法 (注意: 我们在 JList 中画上图像是利用 JList 类所提供的 `setCellRenderer` 这个方法, 与 JComboBox 的 `setRenderer` 名称上有点不一样, 读者请注意)。我们来看下面这个范例:

范例 JComboBox4.java (文件位于随书光盘目录 exam\ch7\JComboBox4.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JComboBox4
6  {
7      String[] s = {"西瓜", "苹果", "草莓", "香蕉", "葡萄"};
8
9      public JComboBox4()
10     {
11         JFrame f = new JFrame("JComboBox");
12         Container contentPane = f.getContentPane();
13
14         JComboBox combo = new JComboBox(s);
15         combo.setBorder(BorderFactory.createTitledBorder("您喜欢吃哪些水果?"));
16         combo.setRenderer(new ACellRenderer());
17         combo.setMaximumRowCount(3);
18
19         contentPane.add(combo);
20         f.pack();
21         f.show();
22         f.addWindowListener(new WindowAdapter() {
23             public void windowClosing(WindowEvent e) {
24                 System.exit(0);
25             }
26         });
27     }
28
29     public static void main(String args[])
30     {
31         new JComboBox4();
32     }
33 }
34
35
36 class ACellRenderer extends JLabel implements ListCellRenderer
37 {
38     ACellRenderer()
39     {
40         setOpaque(true);
41     }
42 }

```

```

43     public Component getListCellRendererComponent(JList list,
44           Object value,
45           int index,
46           boolean isSelected,
47           boolean cellHasFocus)
48     {
49         if (value != null)
50         {
51             setText(value.toString());
52             setIcon(new ImageIcon(".\\icons\\fruit\\"+(index+1)+".jpg"));
53         }
54         if (isSelected) {
55             setBackground(list.getSelectionBackground());
56             setForeground(list.getSelectionForeground());
57         }
58         else {
59             setBackground(list.getBackground());
60             setForeground(list.getForeground());
61         }
62
63         return this;
64     }
65 }

```

⊕ 说明:

- (1) 程序第 16 行, 设置在 JComboBox 中画上图像。在此参数中, 我们产生一个 ACellRenderer 对象, 此对象实作了 ListCellRenderer Interface, 因此可以返回一个 ListCellRenderer 对象作为 setRenderer()方法的参数。
- (2) 程序第 17 行, 设置 JComboBox 一次最多可显示 3 项数据, 其余数据可用 ScrollBar 来选取。
- (3) 程序第 36 行, 类 ACellRenderer 继承 JLabel 并实作 ListCellRenderer。由于我们利用了 JLabel 易于插图的特性, 因此 ACellRenderer 继承了 JLabel, 可让 JComboBox 中的每个项目都被视为是一个 JLabel。
- (4) 程序第 43~64 行, 实作 getListCellRendererComponent()方法。
- (5) 程序第 52 行, 我们所使用的图文件依次是 fruit1.jpg~fruit4.jpg。
- (6) 程序第 54~61 行, 设置选取与取消选取的前景与背景颜色。

⊕ 程序运行结果如图 7-21 所示。



图 7-21

复选框、选项按钮、列表方框、下拉式列表的使用与介绍

本书代码资源文件位于光盘目录 example\ch07\JComboBox5.java

各位读者在运行这个程序时会发现,即使 JComboBox 的选项中有图标,但在选中后图标却不会显示在显示列中。原因是在上面程序第 14 行中,我们以 String Array s 建立 JComboBox:

```
JComboBox combo = new JComboBox(s);
```

String Array s 里面放的只是水果名称,而并没有图标。当我们使用 setRenderer()方法来绘制 JComboBox 时,只会绘制 JComboBox 的选项部分,而最后显示在 JComboBox 上的值还是以 String Array s 为依据。因此 JComboBox 显示列就只会显示文字而已,而不会显示出图形。要解决这个问题,我们必须改变 JComboBox 所传入的参数内容,也就是将原本的 String Array s 改成具有图形的数据项。在此我们是利用 JComboBox(Object[] items)来建立有图像的 JComboBox,我们所传进去的 Object Array 不应该只有文字,而必须连带图标一并传入。我们将上个范例修改如下:

范例 JComboBox5.java (文件位于随书光盘目录 exam\ch7\JComboBox5.java)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JComboBox5
6  {
7      String[] s = {"西瓜", "苹果", "草莓", "香蕉", "葡萄"};
8      ImageIcon[] icons = new ImageIcon[5];
9
10     public JComboBox()
11     {
12         JFrame f = new JFrame("JComboBox");
13         Container contentPane = f.getContentPane();
14         ItemObj[] obj = new ItemObj[5];
15
16         for(int i=0; i < 5; i++)
17         {
18             icons[i] = new ImageIcon(".\\icons\\fruit" + (i+1) + ".jpg");
19             obj[i] = new ItemObj(s[i], icons[i]);
20         }
21
22         JComboBox combo = new JComboBox(obj);
23         combo.setBorder(BorderFactory.createTitledBorder("您喜欢吃哪些水果?"));
24         combo.setRenderer(new ACellRenderer());
25         combo.setMaximumRowCount(3);
26
27         contentPane.add(combo);
28         f.pack();
29         f.show();
30         f.addWindowListener(new WindowAdapter() {
31             public void windowClosing(WindowEvent e) {
32                 System.exit(0);
33             }
34         });
35     }
36
37     public static void main(String args[])
38     {
```

```

39         new JComboBox5();
40     }
41 }
42
43 class ItemObj
44 {
45     String name;
46     ImageIcon icon;
47
48     public ItemObj(String name, ImageIcon icon){
49         this.name = name;
50         this.icon = icon;
51     }
52 }
53
54 class ACellRenderer extends JLabel implements ListCellRenderer
55 {
56     ACellRenderer()
57     {
58         setOpaque(true);
59     }
60
61     public Component getListCellRendererComponent(JList list,
62         Object value,
63         int index,
64         boolean isSelected,
65         boolean cellHasFocus)
66     {
67         if (value != null)
68         {
69             setText(((ItemObj)value).name);
70             setIcon(((ItemObj)value).icon);
71         }
72
73         if (isSelected) {
74             setBackground(list.getSelectionBackground());
75             setForeground(list.getSelectionForeground());
76         }
77         else {
78             setBackground(list.getBackground());
79             setForeground(list.getForeground());
80         }
81
82         return this;
83     }
84 }

```

⊕ 说明:

- (1) 我们在程序第 43~52 行建立一个 ItemObj 类, 里面的内容很简单, 这个类的主要功能是记录水果的名称与相对应的图标。
- (2) 程序第 16~20 行, 将已经建立好的水果名称(String Array s)与水果图标(ImageIcon Array icons) 依次放入 ItemObj Array obj 中。

复选框、选项按钮、列表方框、下拉式列表的使用与介绍

(3) 程序第 22 行, 利用此 ItemObj Array obj 当作是 JComboBox 的传入参数, 构造出 JComboBox。

④ 程序运行结果如图 7-22 所示。



图 7-22

您可以发现, 第一栏显示栏有图标显示出来了。当然您也可以利用 ComboBoxModel 方式来构造出有图标的 JComboBox。我们来看下面这个例子:

范例 JComboBox6.java (文件位于随书光盘目录 exam\ch7\ JComboBox6.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JComboBox5
6  {
7      String[] s = {"西瓜", "苹果", "草莓", "香蕉", "葡萄"};
8      ImageIcon[] icons = new ImageIcon[5];
9
10     public JComboBox5()
11     {
12         JFrame f = new JFrame("JComboBox");
13         Container contentPane = f.getContentPane();
14         for(int i=0; i < 5; i++)
15         {
16             icons[i] = new ImageIcon(".\\icons\\fruit" + (i+1) + ".jpg");
17         }
18         ComboBoxModel mode = new AModel();
19         JComboBox combo = new JComboBox(mode);
20         combo.setBorder(BorderFactory.createTitledBorder("您喜欢吃哪些水果?"));
21         combo.setRenderer(new ACellRenderer());
22         combo.setMaximumRowCount(3);
23
24         contentPane.add(combo);
25         f.pack();
26         f.show();
27         f.addWindowListener(new WindowAdapter() {
28             public void windowClosing(WindowEvent e) {
29                 System.exit(0);
30             }
31         });
32     }
33 }

```

```
32     }
33
34     public static void main(String args[])
35     {
36         new JComboBox5();
37     }
38
39     class AModel extends DefaultComboBoxModel
40     {
41         AModel()
42         {
43             for(int i = 0; i < s.length; i++) {
44                 ItemObj obj = new ItemObj(s[i],icons[i]);
45                 addElement(obj);
46             }
47         }
48     }
49 }
50
51
52 class ItemObj
53 {
54     String name;
55     ImageIcon icon;
56
57     public ItemObj(String name, ImageIcon icon){
58         this.name = name;
59         this.icon = icon;
60     }
61 }
62
63 class ACellRenderer extends JLabel implements ListCellRenderer
64 {
65     ACellRenderer()
66     {
67         setOpaque(true);
68     }
69
70     public Component getListCellRendererComponent(JList list,
71         Object value,
72         int index,
73         boolean isSelected,
74         boolean cellHasFocus)
75     {
76         if (value != null)
77         {
78             setText(((ItemObj)value).name);
79             setIcon(((ItemObj)value).icon);
80         }
81
82         if (isSelected) {
83             setBackground(list.getSelectionBackground());
84             setForeground(list.getSelectionForeground());
85         }
86         else {
```

复选框、选项按钮、列表方框、下拉式列表的使用与介绍

```

87         setBackground(list.getBackground());
88         setForeground(list.getForeground());
89     }
90
91     return this;
92 }
93 }

```

◆ 说明:

- (1) 我们利用 JComboBox(ComboBoxModel aModel)来构造有图标 JComboBox, 因此我们在程序第 39~50 行中, 编写一个继承 DefaultComboBoxModel 的 ComboBoxModel。
 - (2) 在程序第 19 行中, 将构造好的 A Model 传入 JComboBox 中。
- 程序运行结果将同上例。

7-4-4 建立可自行输入的 JComboBox

JComboBox 相对于以前 AWT 的对应组件是 Choice。使用过 Choice 的用户应该很清楚 Choice 有一个令人诟病的地方, 那就是当用户所要的值没有列在项目中时, 用户也无法输入自己想要的值。而在 Swing 中, JComboBox 已经彻底解决这个问题, 而且也相当容易使用, 我们来看下面的例子:

范例 JComboBox7.java (文件位于随书光盘目录 examich7\JComboBox7.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JComboBox7
6  {
7      String[] fontsize = {"12", "14", "16", "18", "20", "22", "24", "26", "28"};
8      String defaultMessage = "请选择或直接输入文字大小! ";
9
10     public JComboBox7()
11     {
12         JFrame f = new JFrame("JComboBox");
13         Container contentPane = f.getContentPane();
14
15         JComboBox combo = new JComboBox(fontsize);
16         combo.setBorder(BorderFactory.createTitledBorder("请选择你要的文字大小"));
17         combo.setEditable(true);
18         ComboBoxEditor editor = combo.getEditor();
19         combo.configureEditor(editor, defaultMessage);
20
21         contentPane.add(combo);
22         f.pack();
23         f.show();
24         f.addWindowListener(new WindowAdapter() {
25             public void windowClosing(WindowEvent e) {
26                 System.exit(0);
27             }
28         });
29     }
30 }

```



```
28         });  
29     }  
30  
31     public static void main(String args[])  
32     {  
33         new JComboBox7();  
34     }  
35 }
```

⊕ 说明：

- (1) 这个程序最重要的地方是第 17 行，将 JComboBox 设成是可编辑的！
- (2) 程序第 18 行，JComboBox 的 getEditor() 方法会返回 ComboBoxEditor 对象。若您察看 Java API Document 可以发现，ComboBoxEditor 是一个 Interface，因此您可以自行实作这个界面，制作出自己想要的 ComboBoxEditor 组件。但通常我们都不需要这么做，因为默认的 ComboBoxEditor 是使用 JTextField，这已经足够应付大部分的情况了。
- (3) 程序第 19 行，JComboBox 的 configureEditor() 方法会初始化 JComboBox 的显示项目。例如这个例子中，我们要 JComboBox 一开始出现“请选择或直接输入文字大小！”这个字符串。

⊕ 程序运行结果如图 7-23 所示。



图 7-23

7-4-5 JComboBox 的事件处理

JComboBox 的事件处理也可分为两种，一种是取得用户选取的项目；另一种是用户在 JComboBox 上自行输入完毕后按下【Enter】键，运行相对应的工作。对于第一种事件的处理，我们使用 ItemListener。对于第二种事件的处理，我们使用 ActionListener。

在下面这个范例中用户可以选取所要的字号，字号的变化会呈现在 JLabel 上，并可以让用户自行输入字体的大小。当用户按下【Enter】键后，若用户输入的值不在选项上时，此输入值会增加至 JComboBox 中，并将输入字体的大小显示在 JLabel 上。范例如下：

复选框、选项按钮、列表方框、下拉式列表的使用与介绍

范例 JComboBox8.java (文件位于随书光盘目录 exam\ch7\JComboBox8.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JComboBox8 implements ItemListener, ActionListener
6  {
7      String[] fontsize = {"12", "14", "16", "18", "20", "22", "24", "26", "28"};
8      String defaultMessage = "请选择或直接输入文字大小! ";
9      Font font = null;
10     JComboBox combo = null;
11     JLabel label = null;
12
13     public JComboBox8()
14     {
15         JFrame f = new JFrame("JComboBox");
16         Container contentPane = f.getContentPane();
17         contentPane.setLayout(new GridLayout(2, 1));
18         label = new JLabel("Swing", JLabel.CENTER);
19         font = new Font("SansSerif", Font.PLAIN, 12);
20         label.setFont(font);
21
22         combo = new JComboBox(fontsize);
23         combo.setBorder(BorderFactory.createTitledBorder("请选择你要的文字大小"));
24         combo.setEditable(true);
25         ComboBoxEditor editor = combo.getEditor();
26         combo.configureEditor(editor, defaultMessage);
27         combo.addItemListener(this);
28         combo.addActionListener(this);
29
30         contentPane.add(label);
31         contentPane.add(combo);
32         f.pack();
33         f.show();
34         f.addWindowListener(new WindowAdapter() {
35             public void windowClosing(WindowEvent e) {
36                 System.exit(0);
37             }
38         });
39     }
40
41     public static void main(String args[])
42     {
43         new JComboBox8();
44     }
45
46     public void actionPerformed(ActionEvent e)
47     {
48         boolean isaddItem = true;
49         int fontsize = 0;
50         String tmp = (String) combo.getSelectedItem();
51
52         try

```

```

53     {
54         fontsize = Integer.parseInt(tmp);
55
56         for (int i=0; i< combo.getItemCount();i++)
57         {
58             if (combo.getItemAt(i).equals(tmp))
59             {
60                 isaddItem = false;
61                 break;
62             }
63         }
64
65         if (isaddItem)
66         {
67             combo.insertItemAt(tmp,0);
68         }
69
70         font = new Font("SansSerif",Font.PLAIN,fontsize);
71         label.setFont(font);
72
73     }catch(NumberFormatException ne){
74         combo.getEditor().setItem("您输入的值不是整数值, 请重新输
75 入!");
76     }
77 }
78
79 public void itemStateChanged(ItemEvent e)
80 {
81     if(e.getStateChange() == ItemEvent.SELECTED)
82     {
83         int fontsize = 0;
84         try
85         {
86             fontsize = Integer.parseInt((String)e.getItem());
87             label.setText("Swing 目前字形大小: "+fontsize);
88         }catch(NumberFormatException ne){
89         }
90     }
91 }
92 }

```

⊕ 说明:

- (1) 程序第 5 行, 由于我们要处理 ItemEvent 与 ActionEvent 事件, 因此在此要先 implements ItemListener,ActionListener。
- (2) 程序第 24 行, 使 JComboBox 变成可编辑状态。
- (3) 程序第 25 行, 取得 ComboBoxEditor。
- (4) 程序第 26 行, 初始化 JComboBox 的显示项目。
- (5) 程序第 27~28 行, 使 JComboBox 可检测 ItemEvent 与 ActionEvent 事件。
- (6) 程序第 46~77 行, ActionListener 界面只有 actionPerformed() 一个方法, 在此实作它。
- (7) 程序第 50 行, JComboBox 的 getSelectedItem() 方法, 可以取得用户所选取的项目。

复选框、选项按钮、列表方框、下拉式列表的使用与介绍

- (8) 程序第 56~63 行, 判断用户所输入的项目是否有重复, 若有重复则不增加到 JComboBox 中。
- (9) 程序第 56 行, 使用 JComboBox 的 getItemCount()方法, 可以取得 JComboBox 中项目的总数。
- (10) 程序第 58 行, 使用 JComboBox 的 getItemAt()方法, 可以取得 JComboBox 中某个项目值。
- (11) 程序第 67 行, 若 JComboBox 中没有这个数值, 则利用 insertItemAt()这个方法, 将数值加到第一列中。
- (12) 程序第 56~63 行, 若所输入的值不是整数, 则在 ComboBoxEditor 中显示“您输入的值不是整数值, 请重新输入!”。
- (13) 程序第 79~91 行, ItemListener 界面只有 itemStateChanged()一个方法, 在此实作它。当用户的选择改变时, 则会在 JLabel 上显示出 Swing 目前字体大小的信息。
- (14) 程序第 89~90 行, 若所输入的值不是整数, 则不作任何的操作。

⊕ 程序运行结果如下:

初始状态, 如图 7-24 所示。



图 7-24

当用户自行输入字号为 36 时, 如图 7-25 所示。



图 7-25

当用户输入的数值不是整数时，如图 7-26 所示。

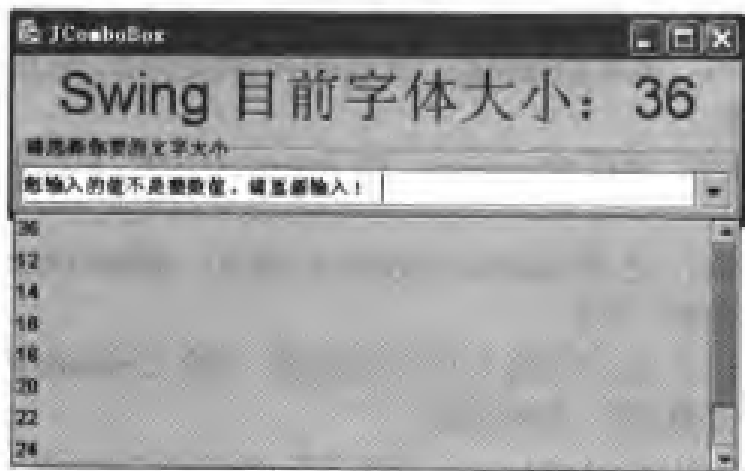


图 7-26

由此图可看出刚刚用户输入的数值 36 已经被增加到 JComboBox 中了。

7-5 本章总结

在此章中，我们先介绍了 JCheckBox 与 JRadioButton 的使用方法、事件处理、以及说明了 JCheckBox 与 JRadioButton 的使用时机。并由范例与讲解中让读者清楚地知道，JCheckBox 与 JRadioButton 除了外观不同外，其余性质几乎完全相同。接下来我们针对 JList 与 JComboBox 进行了解说，这两个组件也有许多相似之处，例如在构造函数或是显示图像部分，均有许多的共同点。在本章中，我们同样也列举了许多时常用到的例子，让读者深刻了解这些组件的用法与事件的处理。

7-6 本章习题

1. JCheckBox 是否可为单选？该如何做到？
2. 试利用事件处理，使 JCheckBox 或 JRadioButton 所选择的项目显示在一个 JLabel 中。
3. 试说明 JList 与 JComboBox 的 DataModel 与 ComboBoxModel 的用处是什么？并分析其关系。
4. 请改写程序 JList8.java，使得项目中具有 Icon 图像。
5. 请改写程序 JComboBox8.java，增加一个 JComboBox，使得用户也可以选择字体样式的功能。

8

表格(Table)的使用与介绍

表格(Table)是我们在设计用户界面(User Interface)时非常重要的一个组件;尤其在我们需要将一堆的统计数据,非常清楚且有条理地显示在用户面前时,表格设计更能显出它的重要。在 Swing 还没出现以前,我们如果要制作一个表格,只能土法炼钢,利用绘图功能一笔一笔地拼凑出我们想要的样子,可说是相当烦人与琐碎。在本章我们将介绍 JTable 强大的功能,并轻松地让您构造与处理 JTable 组件,我们也会举出许多不同的程序范例来供读者参考。

8-1 使用 JTable 组件

表格(Table)是我们在设计用户界面(User Interface) 时非常重要的一个组件; 尤其在我们需要将一堆的统计数据, 非常清楚且有条理的呈现在用户面前时, 表格设计更能显出它的重要。在 Swing 还没出现以前, 我们如果要制作一个表格, 只能土法炼钢, 利用绘图功能一笔一笔地拼凑出我们想要的样子, 可说是相当烦人与琐碎。在 Swing 推出之后, JTable 解决了这样的问题。JTable 可以非常轻松地构造出我们所需要的表格, 并且也提供了一些方法来管理已经设计好的表格内容。

首先, 让我们先来看看 JTable 的类层次结构图:

```
java.lang.Object
    --java.awt.Component
        --java.awt.Container
            --javax.swing.JComponent
                --javax.swing.JTable
```

在使用 JTable 以前, 我们先看一下它的构造函数有哪些, 以及应该如何使用, 如表 8-1 所示。

表 8-1

JTable 的构造函数	
JTable()	建立一个新的 JTable, 并使用系统默认的 Model
JTable(int numRows, int numColumns)	建立一个具有 numRows 行, numColumns 列的空表格, 使用的是 DefaultTableModel
JTable(Object [][] rowData, Object [] columnNames)	建立一个显示二维数组数据的表格, 且可以显示列的名称
JTable(TableModel dm)	建立一个 JTable, 有默认的字段模式以及选择模式, 并设置数据模式
JTable(TableModel dm, TableColumnModel cm)	建立一个 JTable, 设置数据模式与字段模式, 并有默认的选择模式
JTable(TableModel dm, TableColumnModel cm, ListSelectionModel sm)	建立一个 JTable, 设置数据模式、字段模式与选择模式
JTable(Vector rowData, Vector columnNames)	建立一个以 Vector 为输入来源的数据表格, 可显示行的名称

我们先以 Array 构造方式, 说明如何利用 JTable 来建立一个简单的表格:

范例 SimpleTable.java (文件位于随书光盘目录 exam\ch8\SimpleTable.java)

```
1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4  import java.util.*;
5
6  public class SimpleTable {
7      public SimpleTable() {
8
9          JFrame f = new JFrame();
```

```

10      Object[][] playerInfo = {
11          {"阿果", new Integer(66), new Integer(32), new Integer(98), new
            boolean(false)},
12          {"阿瓜", new Integer(85), new Integer(69), new Integer(154), new
            boolean(true)},};
13
14      String[] Names = {"姓名", "语文", "数学", "总分", "及格"};
15
16      JTable table = new JTable(playerInfo, Names);
17      table.setPreferredScrollableViewportSize(new Dimension(550, 30));
18
19      JScrollPane scrollPane = new JScrollPane(table);
20
21      f.getContentPane().add(scrollPane, BorderLayout.CENTER);
22      f.setTitle("Simple Table");
23      f.pack();
24      f.setVisible(true);
25
26      f.addWindowListener(new WindowAdapter() {
27          public void windowClosing(WindowEvent e) {
28              System.exit(0);
29          }
30      });
31  }
32
33  public static void main(String args[]) {
34      SimpleTable b = new SimpleTable();
35  }
36  }

```

说明：

- (1) 程序第 10~12 行为 playerInfo 数组的数据，主要也是我们所建立的表格内的数据。
- (2) 程序第 14 行为 Names 数组的内容，表示表格行的名称。
- (3) 程序第 16 行，我们利用 JTable 类建立了一个以 playerInfo 为内容且以 Names 为行名称的表格；在程序第 17 行，我们以此类所提供的 setPreferredScrollableViewportSize()方法来设置显示表格的窗口大小为 550×30。
- (4) 程序第 19~21 行，我们将表格建立在 Scroll Pane 上，并将 Scroll Pane 加入 Frame 中。
- (5) 程序第 26~30 行，处理关闭窗口口的操作，若您没写这一段，就算您已经关闭了窗口，但程序并不会就此终止。

程序运行结果如图 8-1 所示。

姓名	语文	数学	总分	及格	作弊
阿果	66	32	98	false	
阿瓜	85	69	154	true	

图 8-1

从上图我们可以看出, 表格 (Table) 由两部分组成: 分别是行标题 (Column Header) 与行对象 (Column Object)。Column Header 是在此行的最上端, 代表此行数据的意义, Column Object 则是 Column Header 下面的内容。我们可以利用 JTable 所提供的 getTableHeader() 方法取得行标题。在这个例子中, 我们将 JTable 放在 JScrollPane 中, 这种做法可以将 Column Header 与 Column Object 完整的显示出来, 因为 JScrollPane 会自动取得 Column Header。修改上面程序的第 21 行为:

```
f.getContentPane().add(table, BorderLayout.CENTER);
```

则运行结果您会发现 Column Header 不见了, 如图 8-2 所示。



阿呆	66	32	98	false	false
阿瓜	85	69	154	true	false

图 8-2

如果您不想用 JScrollPane, 要解决这个问题, 您必须将程序修改如下:

```
JTable table = new JTable(p, n);
table.setPreferredScrollableViewportSize(new Dimension(550, 30));
f.getContentPane().add(table.getTableHeader(), BorderLayout.NORTH);
f.getContentPane().add(table, BorderLayout.CENTER);
```

运行结果就会跟之前一样有行标题了。

从上面的运行结果可以发现, 每个字段的宽度都是一样的, 除非您自行拉曳某个列宽。若我们想一开始就设置列宽的值, 可以利用 TableColumn 类所提供的 setPreferredWidth() 方法来设置, 并可利用 JTable 类所提供的 setAutoResizeMode() 方法来设置调整某个列宽时其他列宽的变化情况, 我们看下面这个例子:

范例 SimpleTable2.java (文件位于随书光盘目录 exam\ch8\SimpleTable2.java)

```
1  import javax.swing.*;
2  import javax.swing.table.*;
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.util.*;
6
7  public class SimpleTable2 {
8
9      public SimpleTable2() {
10
11          JFrame f = new JFrame();
12
13          Object[][] p = {
14              {"阿呆", new Integer(66), new Integer(32), new
15              Integer(98), new Boolean(false), new Boolean(false)},
16              {"阿瓜", new Integer(85), new Integer(69), new
17              Integer(154), new Boolean(true), new Boolean(false)};
18
19          String[] n = {"姓名", "语文", "数学", "总分", "及格", "作弊"};
20
21          TableColumn column = null;
```

```

22         JTable table = new JTable(p, n);
23         table.setPreferredScrollableViewportSize(new Dimension(550, 30));
24         table.setAutoResizeMode(JTable.AUTO_RESIZE_SUBSEQUENT_COLUMNS);
25
26         for (int i=0; i<6 ;i++)
27         {
28             column = table.getColumnModel().getColumn(i);
29             if ((i % 2) == 0)
30                 column.setPreferredWidth(150);
31             else
32                 column.setPreferredWidth(50);
33         }
34
35         //Create the scroll pane and add the table to it.
36         JScrollPane scrollPane = new JScrollPane(table);
37
38         //Add the scroll pane to this window.
39         f.getContentPane().add(scrollPane, BorderLayout.CENTER);
40         f.setTitle("Simple Table");
41         f.pack();
42         f.setVisible(true);
43
44         f.addWindowListener(new WindowAdapter() {
45             public void windowClosing(WindowEvent e) {
46                 System.exit(0);
47             }
48         });
49
50     }
51
52     public static void main(String args[]) {
53
54         new SimpleTable2();
55     }
56 }

```

⊕ 说明:

- (1) 程序第 24 行, 设置当调整某个列宽时, 其他列宽的变化情形。JTable 提供了 5 个参数可以使用, 分别是:

AUTO_RESIZE_SUBSEQUENT_COLUMNS: 当调整某一列宽时, 此字段之后的所有字段列宽都会跟着一起变动。此为系统默认值。

AUTO_RESIZE_ALL_COLUMNS: 当调整某一列宽时, 此表格上所有字段的列宽都会跟着一起变动。

AUTO_RESIZE_OFF: 当调整某一列宽时, 此表格上所有字段列宽都不会跟着改变。

AUTO_RESIZE_NEXT_COLUMN: 当调整某一列宽时, 此字段的下一个字段的列宽会跟着改变, 其余均不会变。

AUTO_RESIZE_LAST_COLUMN: 当调整某一列宽时, 最后一个字段的列宽会跟着改变, 其余均不会变。

- (2) 程序第 29~32 行, 设置偶数列的列宽为 150 个 pixel, 奇数列的列宽为 50 个 pixel。
- (3) 程序第 28 行, 利用 `JTable` 中的 `getColumnModel()` 方法取得 `TableColumnModel` 对象; 再利用 `TableColumnModel` 界面所定义的 `getColumn()` 方法取得 `TableColumn` 对象, 利用此对象的 `setPreferredWidth()` 方法就可以控制字段的宽度。

◆ 程序运行结果如图 8-3 所示。

姓名	语文	数学	英语	及格	优秀
小明	88	92	88	false	false
阿凡	85	88	154	true	false

图 8-3

由上面的范例可知, 利用 Swing 来构造一个表格其实是很简单的, 只要您利用 `Vector` 或 `Array` 来作为我们表格的数据输入, 将 `Vector` 或 `Array` 的内容填入 `JTable` 中, 一个基本的表格就产生了。不过, 虽然利用 `JTable (Object[][] rowData, Object[] columnNames)` 以及 `JTable (Vector rowData, Vector columnNames)` 构造函数来构造 `JTable` 很方便, 但却有些缺点。例如上例中, 我们表格中的每个字段 (`Cell`) 一开始都是默认为可修改的, 用户因此可能修改到我们的数据; 其次, 表格中每个字段 (`Cell`) 中的数据类型将会被视为同一种。在我们的例子中, 数据类型皆被显示为 `String` 的类型, 因此, 原来数据类型声明为 `Boolean` 的数据会以 `String` 的形式出现而不是以检查框 (`Check Box`) 的形式出现。

除此之外, 如果我们所要显示的数据是不固定的, 或是随情况而变的, 例如同样是一份成绩单, 老师与学生所看到的表格就不会一样, 显示的外观或操作模式可能也不相同。为了应对这些复杂的情况, 上面简单的构造方式已不敷使用, Swing 提供了各种 `Model` (如 `TableModel`、`TableColumnModel` 与 `ListSelectionModel`) 来解决上述的不便, 以增加我们设计表格的弹性。我们接下来就先对 `TableModel` 来作介绍。

8-2 TableModel

`TableModel` 类本身是一个 `Interface`, 在这个 `Interface` 里面定义了若干的方法: 包括了存取表格字段 (`Cell`) 的内容、计算表格的列数等等的基本存取操作, 让设计者可以简单地利用 `TableModel` 来实现他所想要的表格。`TableModel` 界面放在 `javax.swing.table` package 中, 这个 package 定义了许多 `JTable` 会用到的 `Model`, 读者可利用 Java API 文件找到这个 package, 并由此 package 找到各类或界面所定义的方法。表 8-2 是 `TableModel` 所定义的方法。

表 8-2

TableModel 方法	
void	addTableModelListener(<code>TableModelListener l</code>) 使表格具有处理 <code>TableModelEvent</code> 的能力。当表格的 <code>Table Model</code> 有所变化时, 会发出 <code>TableModel-Event</code> 事件信息

线上表

TableModel 方法	
Class	getColumnClass(int columnIndex) 返回字段数据类型的类名称
int	getColumnCount() 返回字段（行）数量
String	getColumnName(int columnIndex) 返回字段名称
int	getRowCount() 返回数据列数量
Object	getValueAt(int rowIndex, int columnIndex) 返回某个 Cell 中的值
boolean	isCellEditable(int rowIndex, int columnIndex) 返回 Cell 是否可编辑, true 为可编辑
void	removeTableModelListener(TableModelListener l) 从 TableModelListener 中移除一个 listener
void	setValueAt(Object aValue, int rowIndex, int columnIndex) 设置某个 Cell (rowIndex, columnIndex)的值

由于 TableModel 本身是一个 Interface, 因此若要直接实现此界面来建立表格并不是件轻松的事。幸好 Java 提供了两个类分别实现了这个界面, 一个是 AbstractTableModel 抽象类, 一个是 DefaultTableModel 实体类。前者实现了大部分的 TableModel 方法, 让用户可以很有弹性的构造出自己的表格模式; 后者继承前者类, 是 Java 默认的表格模式。这三者的关系如图 8-4 所示。

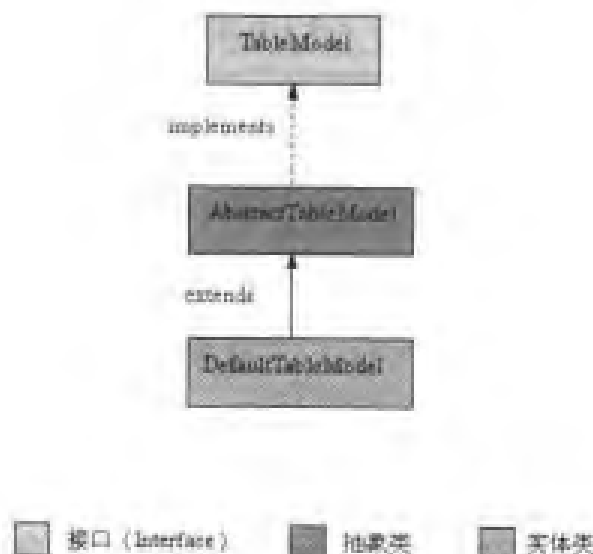


图 8-4

这种类关系跟上一章所介绍的 JList 性质非常像, 读者可回想我们在第 1 章所谈论到的 MVC 结构, 就能明白这些组件为何这么设计了。下面我们就来看如何利用 AbstractTableModel 构造出不同的表格内容。

8-3 AbstractTableModel

Java 提供的 AbstractTableModel 是一个抽象类, 这个类帮我们实现大部分的 TableModel 方法, 除了 `getRowCount()`、`getColumnCount()`、`getValueAt()` 这三个方法外。因此我们的主要任务就是要去实现这三个方法。利用这个抽象类就可以设计出不同格式的表格。我们先来看看 AbstractTableModel 这个类所提供的方法, 如表 8-3 所示。

表 8-3

AbstractTableModel 方法	
void	<code>addTableModelListener(TableModelListener l)</code> 使表格具有处理 TableModelEvent 的能力。当表格的 Table Model 有所变化时, 会发出 TableModelEvent 事件信息
int	寻找在行名称中是否含有 columnName 这个项目。若有, 则返回其所在行的位置; 反之则返回 -1 表示找不到
void	<code>fireTableCellUpdated(int row, int column)</code> 通知所有的 listener 在这个表格中的 (row, column) 字段的内容已经改变了
void	<code>fireTableChanged(TableModelEvent e)</code> 将所收的事件通知传送给所有在这个 table model 中注册过的 TableModelListeners
void	<code>fireTableDataChanged()</code> 通知所有的 listener 在这个表格中列的内容已经改变了。列的数目可能已经改变了, 因此 JTable 可能需要重新显示此表格的结构
void	<code>fireTableRowsDeleted(int firstrow, int lastrow)</code> 通知所有的 listener 在这个表格中第 firstrow 行至 lastrow 行已经被删除了
void	<code>fireTableRowsInserted(int firstrow, int lastrow)</code> 通知所有的 listener 在这个表格中第 firstrow 行至 lastrow 行已经被加入了
void	通知所有的 listener 在这个表格中第 firstrow 行至 lastrow 行已经被修改了
void	<code>fireTableStructureChanged(int firstrow, int lastrow)</code> 通知所有的 listener 在这个表格的结构已经改变了。行的数目、名称以及数据类型都可能已经改变了
Class	<code>getColumnClass(int columnIndex)</code> 返回字段数据类型的类名称
String	<code>getColumnName(int column)</code> 若没有设置列标题名称则返回默认值, 依次为 A, B, C, ... Z, AA, AB, ...; 若无此 column, 则返回一个空的 String
public EventListener []	<code>getListeners(Class listenerType)</code> 返回所有在这个 table model 上所建立的 listener 中符合 listenerType 形式的 listener, 并以数组形式返回
boolean	<code>isCellEditable(int rowIndex, int columnIndex)</code> 若表格(rowIndex, columnIndex)为可修改的字段, 则返回 true, 否则返回 false。默认值为 false
void	<code>removeTableModelListener(TableModelListener l)</code> 从 TableModelListener 中移除一个 listener
void	<code>setValueAt(Object aValue, int rowIndex, int columnIndex)</code> 设置某个 Cell (rowIndex, columnIndex)的值

若您仔细比较 `TableModel` 所定义的方法与上述 `AbstractTableModel` 所提供的方法, 您会发现, `AbstractTableModel` 抽象类并没有实现 `getRowCount()`、`getColumnCount()`、`getValueAt()` 这三个方法, 这也就是为什么我们要去实现这三个方法的原因。下面我们来看如何使用 `AbstractTableModel` 实现出自己想要的表格模式:

范例 `TableModel1.java`(文件位于随书光盘目录 `exam\ch8\TableModel1.java`)

```

1  import javax.swing.table.AbstractTableModel;
2  import javax.swing.*;
3  import java.awt.*;
4  import java.awt.event.*;
5
6  public class TableModel1 {
7
8      public TableModel1 () {
9
10         JFrame f = new JFrame();
11         MyTable mt=new MyTable();
12         JTable t=new JTable(mt);
13
14         t.setPreferredScrollableViewportSize(new Dimension(550, 30));
15         JScrollPane s = new JScrollPane(t);
16         f.getContentPane().add(s, BorderLayout.CENTER);
17
18         f.setTitle("JTable1");
19         f.pack();
20         f.setVisible(true);
21
22         f.addWindowListener(new WindowAdapter() {
23             public void windowClosing(WindowEvent e) {
24                 System.exit(0);
25             }
26         });
27     }
28
29     public static void main(String args[]) {
30         new TableModel1 ();
31     }
32 }
33
34 class MyTable extends AbstractTableModel
35 {
36     Object[][] p = {"阿呆", new Integer(66), new Integer(32), new Integer(98),
37                     new Boolean(false),new Boolean(false)},
38                     {"阿瓜", new Integer(85), new Integer(69), new
39                     Integer(154), new Boolean(true),new Boolean(false)};
40
41     String[] n = {"姓名", "语文", "数学", "总分", "及格", "作弊"};
42
43     public int getColumnCount() {
44         return n.length;
45     }
46
47     public int getRowCount() {

```

```

48         return p.length;
49     }
50
51     public Object getValueAt(int row, int col) {
52         return p[row][col];
53     }
54 }

```

⊕ 说明：

- (1) 程序第 11 行，由于我们的 MyTable 类继承了 AbstractTableModel 并且实现了 getColumnCount()、getRowCount()、getValueAt() 方法，因此我们可以通过 MyTable 来产生 TableModel 的实体。
- (2) 在程序第 12 行中，我们利用 MyTable 建立表格模式。
- (3) 程序第 34~53 行，为 MyTable 类的方法。

⊕ 图 8-5 为此程序的运行结果：



A	B	C	D	E	F
阿呆	88	32	98	false	false
阿瓜	85	68	154	true	false

图 8-5

咦！？怎么跟我们上一节中的例子好像有那么点不一样？发现了吗，每一行的名称都换成了 A、B、C 等英文字母来表示，这是因为我们没有设置字段名称。没关系，只要我们在 MyTable 类中改写 getColumnName() 这个方法，就可以把行标题显示出来。

```

public String getColumnName(int col) {
    return n[col];
}

```

再来看看运行结果是否变了，如图 8-6 所示。



姓名	语文	数学	总分	及格	作对
阿呆	88	32	98	false	false
阿瓜	85	68	154	true	false

图 8-6

在本节的一开始我们提到过，getRowCount()、getColumnCount()、getValueAt() 三个方法是在我们继承 AbstractTableModel 类时所必须实现的方法。但如果我们“不小心”忘了实作怎么办？此时您在编译此程序时就会出现 error 的信息。例如若我们没有编写 getColumnCount() 这个方法，聪明的 Java 会以下面的信息提醒您，如图 8-7 所示。

```

tableModel1.java:15: MyTable should be declared abstract: it does not define get
ColumnCount(). in java.awt.table.TableModel
class MyTable extends AbstractTableModel
^
1 error

```

图 8-7

从上面的运行结果我们发现,表格内的数据类型不论是 String、Int 亦或是 Boolean 类型,都以 String 的类型显示。例如在“及格”字段中,原本的数据是以 Boolean 类型来表示,但显示在 JTable 上时便转换成字符串形式。若想要使表格能够显示出不同的数据类型,我们需要在 MyTable 类中覆写(Override) getColumnClass() 方法。这个方法可以让我们分辨出表格中每一行的数据类型,并将此类型作适当的显示,此方法的实作如下:

```
public Class getColumnClass(int columnIndex) {
    return getValueAt(0, columnIndex).getClass();
}
```

运行结果如图 8-8 所示。

姓名	语文	数学	总分	及格	作弊
阿呆	66	32	98	<input type="checkbox"/>	<input type="checkbox"/>
阿成	65	69	134	<input type="checkbox"/>	<input type="checkbox"/>

图 8-8

经过 getColumnClass() 方法的处理之后,我们就能判断出每行的数据类型是什么。在 JTable 中,如果数据类型为 String,一律靠左对齐;如果数据类型为 Integer,一律靠右对齐;如果数据类型为 Boolean,则以 Check Box 且置中表示之。例如“姓名”为 String 类型,因此靠左显示;“语文”、“数学”以及“总分”这几列为 Integer 类型,因此靠右显示;而“及格”及“作弊”都是 Boolean 的类型,因此以 Check Box 显示。

综上所述,我们来看一个较复杂的例子:

此例子的表格模式管理了数学老师与学生阿呆的显示内容,当按下“数学老师”按钮时即会显示出所有学习数学课程的学生成绩,但数学老师不能看到学生其他课程的成绩。当用户按下“学生阿呆”按钮时会显示出学生阿呆所有修课的成绩,但他看不到其他同学的成绩。读者若会利用 Java 存取数据库的数据,可利用这个例子的概念做出更多的变化。

范例 TableModel2.java(文件位于随书光盘目录 exam\ch8\TableModel2.java)

```
1  import javax.swing.table.AbstractTableModel;
2  import javax.swing.*;
3  import java.awt.*;
4  import java.awt.event.*;
5
6  public class TableModel2 implements ActionListener{
7
8      JTable t = null;
9
10     public TableModel2() {
11
12         JFrame f = new JFrame("DataModel");
13         JButton b1 = new JButton("数学老师");
14         b1.addActionListener(this);
15         JButton b2 = new JButton("学生阿呆");
16         b2.addActionListener(this);
17         JPanel panel = new JPanel();
18         panel.add(b1);
19         panel.add(b2);
20     }
```



```
21         t=new JTable(new MyTable(1));
22         t.setPreferredScrollableViewportSize(new Dimension(550, 30));
23         JScrollPane s = new JScrollPane(t);
24
25         f.getContentPane().add(panel, BorderLayout.NORTH);
26         f.getContentPane().add(s, BorderLayout.CENTER);
27         f.pack();
28         f.setVisible(true);
29
30         f.addWindowListener(new WindowAdapter() {
31             public void windowClosing(WindowEvent e) {
32                 System.exit(0);
33             }
34         });
35     }
36
37     public void actionPerformed(ActionEvent e)
38     {
39         if (e.getActionCommand().equals("学生阿呆"))
40             t.setModel(new MyTable(1));
41         if (e.getActionCommand().equals("数学老师"))
42             t.setModel(new MyTable(2));
43         t.revalidate();
44     }
45
46     public static void main(String args[]) {
47
48         new TableModel2();
49     }
50 }
51
52 class MyTable extends AbstractTableModel{
53
54     Object[][] p1 = {
55         {"阿呆", "1234",new Integer(66),new Integer(50), new Integer(116),
56          new Boolean(false),new Boolean(false)}};
57
58     String[] n1 = {"姓名","学号","语文","数学","总分","及格","作弊"};
59
60     Object[][] p2 = {
61         {"阿呆", "1234",new Integer(50), new Boolean(false),new Boolean(
62          false),"01234"},
63         {"阿瓜", "1235",new Integer(75), new Boolean(true),new Boolean(
64          false),"05678"}};
65
66     String[] n2 = {"姓名","学号","数学","及格","作弊","电话"};
67
68     //model 为 1 表示显示出阿呆学生的表格, model 为 2 则显示数学老师的表格
69     int model = 1;
70
71     public MyTable(int i){
72         model = i;
73     }
74
75     public int getColumnCount() {
```

```

73         if(model ==1)
74             return n1.length;
75         else
76             return n2.length;
77     }
78
79     public int getRowCount() {
80         if(model ==1)
81             return p1.length;
82         else
83             return p2.length;
84     }
85
86     public Object getValueAt(int row, int col) {
87         if(model == 1)
88             return p1[row][col];
89         else
90             return p2[row][col];
91     }
92
93     public String getColumnName(int col) {
94         if(model ==1)
95             return n1[col];
96         else
97             return n2[col];
98     }
99
100    public Class getColumnClass(int c) {
101        return getValueAt(0, c).getClass();
102    }
103 }

```

说明:

- (1) 我们更改上个例子, 在这个例子中的 MyTable 类除了实现 getColumnCount()、getRowCount()、getValueAt()方法外, 还覆写 (Override) 了 getColumnName()与 getColumnClass()这两个方法, 使表格上的字段名称与字段中的数据类型能正确的显示出来。
- (2) 程序第 37~44 行, 当按下“学生阿呆”按钮时, 设置表格模式为 model 1; 当按下“数学老师”按钮时, 设置表格模式为 model 2。
- (3) 程序第 66 行, 设置程序一开始运行时先显示学生阿呆的成绩。

程序运行结果如图 8-9 所示。

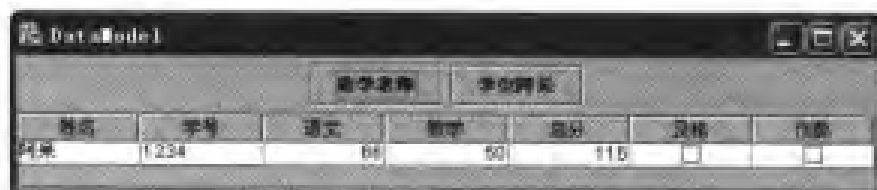


图 8-9

当按下“数学老师”按钮时，如图 8-10 所示。

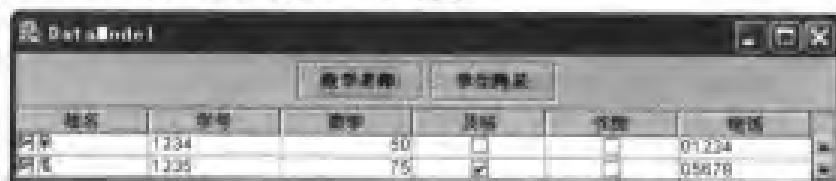


图 8-10

由上图可看出数学老师只能看到学生的数学成绩，而学生阿呆也只能看到他自己的成绩。若读者亲自运行此程序时可以发现表格中的每项数值是不能更改的，这跟之前利用 `JTable(Object [][] rowData, Object [] columnNames)` 构造函数所构造出来的表格特性刚好相反。不过读者可覆写(Override) `AbstractTableModel` 中的 `isCellEditable()` 方法，来设置表格中的字段是否为可编辑。

从上面的例子，我们归纳出在 `JTable` 中的每一行，依数据类型的不同可以有下列的排列显示方式：

- (1) Boolean: 以 Check Box 表示。
- (2) Number: 以 `JLabel` 表示，文字向右排列。
- (3) ImageIcon: 以图形 `JLabel` 表示，图形置于 `JLabel` 的中央。
- (4) Object: 以 `JLabel` 显示对象的 `String` 内容，文字向左排列。

8-4 TableColumnModel

`TableColumnModel` 本身是一个 `Interface`，里面定义了许多与表格的“列(行)”有关的方法，例如增加列，删除列，设置与取得“列”的相关信息等等。通常我们不会直接实现 `TableColumnModel` 界面，而是会利用 `JTable` 的 `getColumnModel()` 方法取得 `TableColumnModel` 对象，再利用此对象对字段做设置。举例来说，如果我们想设计的表格是包括有下拉式列表的 `Combo Box`，我们就能利用 `TableColumnModel` 来达到这样的效果。

我们先看下面的例子：

范例 `ColumnModelTest.java` 光盘目录 `exam\ch8\ ColumnModelTest.java`

```

1  import javax.swing.table.AbstractTableModel;
2  import javax.swing.*;
3  import java.awt.*;
4  import java.awt.event.*;
5
6  public class ColumnModelTest {
7
8      public ColumnModelTest () {
9
10         JFrame f = new JFrame();
11         MyTable mt=new MyTable();
12         JTable t=new JTable(mt);
13
14         JComboBox c = new JComboBox();
15         c.addItem("Taipei");
16         c.addItem("ChiaYi");

```

```

17         c.addItem("HsinChu");
18         t.getColumnModel().getColumn(1).setCellEditor(new
           DefaultCellEditor(c));
19
20         t.setPreferredScrollableViewportSize(new Dimension(550, 30));
21         JScrollPane s = new JScrollPane(t);
22
23         f.getContentPane().add(s, BorderLayout.CENTER);
24         f.setTitle("ColumnModelTest ");
25         f.pack();
26         f.setVisible(true);
27
28         f.addWindowListener(new WindowAdapter() {
29             public void windowClosing(WindowEvent e) {
30                 System.exit(0);
31             }
32         });
33     }
34
35     public static void main(String args[]) {
36         new ColumnModelTest ();
37     }
38 }
39
40 class MyTable extends AbstractTableModel{
41
42     Object[][] p = {
43         {"阿果", "Taipei", new Integer(66), new Integer(32),
44          new Integer(98), new Boolean(false), new Boolean(false)},
45         {"阿瓜", "ChiaYi", new Integer(85),
46          new Integer(69), new Integer(154), new Boolean(true), new
47          Boolean(false)}, };
48
49     String[] n = {"姓名", "居住地", "语文", "数学", "总分", "及格", "作弊"};
50
51     public int getColumnCount() {
52         return n.length;
53     }
54
55     public int getRowCount() {
56         return p.length;
57     }
58
59     public String getColumnName(int col) {
60         return n[col];
61     }
62
63     public Object getValueAt(int row, int col) {
64         return p[row][col];
65     }
66
67     public Class getColumnClass(int c) {
68         return getValueAt(0, c).getClass();
69     }

```

◆ 说明:

- (1) 程序第 11 行, 由于我们的 MyTable 类继承了 AbstractTableModel 并且实现了 getColumnCount()、getRowCount()、getValueAt()方法。因此我们可以通过 MyTable 来产生 TableModel 的实体。
- (2) 程序第 12 行, 我们利用 MyTable 来建立 JTable。
- (3) 程序第 14 行, 建立一个 JComboBox 的对象。
- (4) 程序第 15~17 行, 我们在新建立的 JComboBox 对象里新增三个项目, 分别为 Taipei、ChiaYi、HsinChu。
- (5) 程序第 18 行, 我们利用 JTable 所提供的 getTableColumnModel()方法取得 TableColumnModel 对象, 再由 TableColumnModel 类所提供的 getColumn()方法取得 TableColumn 对象, TableColumn 类可针对表格中的每一行做具体的设置, 例如设置字段的宽度、某行的标头、设置输入较复杂的数据类型等等。在这里, 我们利用 TableColumn 类所提供的 setCellEditor() 方法, 将 JComboBox 作为第二行的默认编辑组件。
- (6) 程序第 40~68 行为 MyTable 类的方法。

◆ 程序运行结果如图 8-11 所示。



姓名	居住地址	语文	数学	总分	及格	性别
阿基	Taipei	60	32	90	<input type="checkbox"/>	
阿成	ChiaYi	85	88	104	<input checked="" type="checkbox"/>	

图 8-11

读者运行此程序可以发现, 利用继承 AbstractTableModel 抽象类所产生的 JTable 内容是不能被修改的。那如果想要让用户可以修改表格中的某一个字段, 例如勾选 Check Box 或是直接修改某个字段的数字, 该怎么做呢? 很简单, 只要我们在范例中的 MyTable 类中覆写 AbstractTableModel 抽象类中的 isCellEditable()方法即可。下面即是 isCellEditable()的实现:

```
public boolean isCellEditable(int rowIndex, int columnIndex) {  
    return true;  
}
```

在 isCellEditable()中, 我们只有一行简单的程序代码: return true, 意思是将我们表格内的每个 Cell 都变成可修改。但仅仅修改这个程序代码还不行, 您可以发现虽然表格现在变成可以修改了, 但更改完之后如果按下【Enter】键, 内容马上恢复成原有的值。解决的方式是覆写 AbstractTableModel 抽象类中的 setValueAt()方法, 这个方法主要是让我们将改过的值存入表格中, 如下所示:

```
public void setValueAt(Object value, int row, int col) {  
    p[row][col] = value;  
    fireTableCellUpdated(row, col);  
}
```

其中 value 为我们所更改的值，我们将 value 存入 `p[row][col]` 中，并且调用 `fireTableCellUpdated()` 函数来告诉我们的系统表格已经做了更改，关于这一部分，我们在本章后面会再对事件处理作详细地介绍，在此范例中有没有加入 `fireTableCellUpdated()` 方法对运行结果不会造成影响。我们先来看看在加入这个方法之后，表格发生了什么变化，如图 8-12 所示。



图 8-12

在上图中不只 Combo Box 已经可以正常地运作，连 Check Box 和其他字段也能作修改了（如我们将阿呆成绩设为及格，将阿瓜语文成绩设为 60）。读者可将 `isCellEditable()` 与 `setValueAt()` 方法加入程序 `ColumnModelTest.java` 中，并试试其效果。

8-5 SelectionModel

表格的选择模式是依据我们前章所讲的 `ListSelectionModel` 而来，因此它的操作模式与事件处理跟 `JList` 没什么分别。我们稍微复习一下 `ListSelectionModel` 这个 Interface，它包含 3 个常数值，如下：

```
static int SINGLE_SELECTION
static int SINGLE_INTERVAL_SELECTION
static int MULTIPLE_INTERVAL_SELECTION
```

分别可让用户作单一选择、连续区间选择与多重选择。当用户作后面两个模式的操作时，应配合【Shift】键或【Ctrl】键。

要使用 `ListSelectionModel` 可利用 `JTable` 的 `getSelectionMode()` 方法取得 `ListSelectionModel` 对象，再利用 `ListSelectionModel` 界面所定义的 `setSelectionMode()` 来设置选择模式。

如同 `JList` 一般，当用户对表格作数据域位的选取时会产生 `ListSelectionEvent` 事件，要处理这个事件就必须实现 `ListSelectionListener` 这个界面，此界面定义了一个方法，那就是 `valueChanged()`。

我们来看下面的例子，用户可在按钮上选择哪种选择模式，当用户选取表格数据时，程序会将用户选取的数据显示在表格下面的 `JLabel` 中。

范例 SelectionModelDemo.java (文件位于随书光盘目录
exam\ch8\SelectionModelDemo.java)

```
1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4 import javax.swing.event.*;
5
6 public class SelectionModelDemo implements ActionListener, ListSelectionL-
7 istener
```

```
7      {
8          JTable table = null;
9          ListSelectionModel selectionMode = null;
10         JLabel label = null; //显示用户选取表格训之用
11
12         public SelectionModelDemo()
13         {
14             JFrame f = new JFrame();
15             String[] name = {"字段 1","字段 2","字段 3","字段 4","字段 5"};
16             String[][] data = new String[5][5];
17             int value =1;
18             for(int i=0; i<data.length; i++)
19             {
20                 for(int j=0; j<data[i].length ; j++)
21                     data[i][j] = String.valueOf(value++);
22             }
23
24             table=new JTable(data,name);
25             table.setPreferredScrollableViewportSize(new Dimension(400, 80));
26             table.setCellSelectionEnabled(true);
27             selectionMode = table.getSelectionModel();
28             selectionMode.addListSelectionListener(this);
29             JScrollPane s = new JScrollPane(table);
30
31             JPanel panel = new JPanel();
32             JButton b = new JButton("单一选择");
33             panel.add(b);
34             b.addActionListener(this);
35             b = new JButton("连续区间选择");
36             panel.add(b);
37             b.addActionListener(this);
38             b = new JButton("多重选择");
39             panel.add(b);
40             b.addActionListener(this);
41
42             label = new JLabel("您选取: ");
43
44             Container contentPane = f.getContentPane();
45             contentPane.add(panel, BorderLayout.NORTH);
46             contentPane.add(s, BorderLayout.CENTER);
47             contentPane.add(label, BorderLayout.SOUTH);
48
49             f.setTitle("SelectionModelDemo");
50             f.pack();
51             f.setVisible(true);
52
53             f.addWindowListener(new WindowAdapter() {
54                 public void windowClosing(WindowEvent e) {
55                     System.exit(0);
56                 }
57             });
58         }
59
60         public void actionPerformed(ActionEvent e)
61         {
```

```

62         if(e.getActionCommand().equals("单一选择"))
63             selectionMode.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
64         if(e.getActionCommand().equals("连续区间选择"))
65             selectionMode.setSelectionMode(
66                 ListSelectionModel.SINGLE_INTERVAL_SELECTION);
67         if(e.getActionCommand().equals("多重选择"))
68             selectionMode.setSelectionMode(
69                 ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
70         table.revalidate();
71     }
72
73
74     public void valueChanged(ListSelectionEvent e1)
75     {
76         String tempString = "";
77         int[] rows = table.getSelectedRows();
78         int[] columns = table.getSelectedColumns();
79
80         for (int i=0; i<rows.length; i++) {
81             for (int j=0; j<columns.length; j++)
82                 tempString = tempString+" "+(String)table.getValueAt(rows-
83                     [i], columns[j]);
84             label.setText("您选取: "+tempString);
85         }
86
87         public static void main(String args[]) {
88             new SelectionModelDemo();
89         }
90     }

```

◆ 说明:

- (1) 在此范例中, 我们要处理 `ActionEvent` 与 `ListSelectionEvent`, 因此在程序第 6 行中我们要实现 `ActionListener` 与 `ListSelectionListener` 界面。而 `ListSelectionEvent` 是属于 `Swing` 的事件, 因此程序第 4 行我们要 `import javax.swing.event package` 进来。
- (2) 程序第 26 行, `setCellSelectionEnabled(true)` 方法使得表格的选取是以 `Cell` 为单位, 而不是以列为单位。若您没有写此行, 则在选取表格数据时都是以整列为单位。
- (3) 程序第 27 行, 取得 `table` 的 `ListSelectionModel`。
- (4) 程序第 60~72 行, 处理按钮事件。利用 `ListSelectionModel` 界面所定义的 `setSelectionMode()` 方法来设置表格选取模式。
- (5) 程序第 74~85 行, 当用户选取表格数据时会触发 `ListSelectionEvent`, 我们实现 `ListSelectionListener` 界面来处理这一事件。`ListSelectionListener` 界面只定义了一个方法, 那就是 `valueChanged()`。
- (6) 程序第 77~78 行, `JTable` 的 `getSelectedRows()` 与 `getSelectedColumns()` 方法会返回已选取表格 `Cell` 的 `index Array` 数据。
- (7) 程序第 81 行, `JTable` 的 `getValueAt()` 方法会返回某列某行的 `cell` 数据, 返回值是 `Object` 数据类型, 因此我们要自行转成 `String` 数据类型。

程序运行结果如图 8-13 所示。



图 8-13

表格一开始为“多重选择”状态，因此您可以使用【Ctrl】键或【Shift】键来选取表格中的多个 Cell 数据。当您按下“单一选择”按钮时，【Ctrl】键或【Shift】键就不会有作用，当您按下“连续区间选择”按钮时，【Ctrl】键不会有作用。例如当我们点选“多重选择”按钮时，可以选取表格数据如图 8-14 所示的状态：

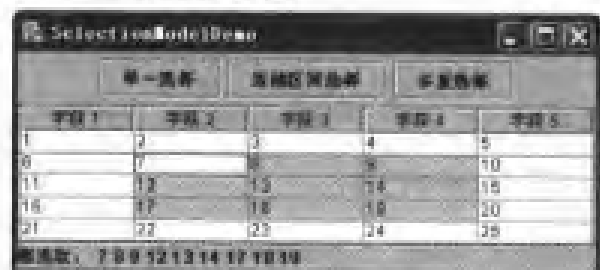


图 8-14

8-6 DefaultTableModel

我们曾在 8-2 节曾提到过 DefaultTableModel 类，并说明了此类是继承 AbstractTableModel 抽象类而来，且实现了 getColumnCount()、getRowCount()与 getValueAt()这三种方法。因此在使用上，DefaultTableModel 比 AbstractTableModel 要来得简单许多，也较常被拿来使用。DefaultTableModel 内部是使用 Vector 来处理表格的数据，若您所要显示的表格格式是比较单纯的变化，笔者建议使用 DefaultTableModel 类会很方便也简单许多。若您所要显示的数据模式非常复杂，例如我们所举的成绩表格外加学生选课信息等，像这类的表格通常显示的信息会因人而异，因此使用 AbstractTableModel 会比较容易设计些。

表 8-4 是 DefaultTableModel 的构造函数：

表 8-4

DefaultTableModel 的构造函数

DefaultTableModel()

建立一个 DefaultTableModel，里面没有任何数据

DefaultTableModel(int numRows, int numColumns)

建立一个指定行列数的 DefaultTableModel

DefaultTableModel(Object[][] data, Object[] columnNames)

建立一个 DefaultTableModel，输入数据格式为 Object Array。系统会自动调用 setDataVector()方法来设置数据

续上表

DefaultTableModel 的构造函数

DefaultTableModel(Object[] columnNames, int numRows)

建立一个 DefaultTableModel, 并具有 Column Header 名称与行数信息

DefaultTableModel(Vector columnNames, int numRows)

建立一个 DefaultTableModel, 并具有 Column Header 名称与行数信息

DefaultTableModel(Vector data, Vector columnNames)

建立一个 DefaultTableModel, 输入数据格式为 Vector。系统会自动调用 setDataVector()方法来设置数据

DefaultTableModel 类提供了很多好用的方法, 如之前我们谈论过的 getColumnCount()、getRowCount()、getValueAt()、isCellEditable()、setValueAt()等方法, 均可直接使用。且 DefaultTableModel 也提供了 addColumn()与 addRow()等方法, 可让我们随时增加表格的数据。下面我们就举一个动态增加表格字段的例子:

范例 AddRemoveCells.java(文件位于随书光盘目录

exam\ch8\AddRemoveCells.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import java.util.Vector;
4  import javax.swing.*;
5  import javax.swing.event.*;
6  import javax.swing.table.*;
7
8  public class AddRemoveCells implements ActionListener
9  {
10     JTable table = null;
11     DefaultTableModel defaultModel = null;
12
13     public AddRemoveCells()
14     {
15         JFrame f = new JFrame();
16         String[] name = {"字段 1", "字段 2", "字段 3", "字段 4", "字段 5"};
17         String[][] data = new String[5][5];
18         int value = 1;
19         for(int i=0; i<data.length; i++)
20         {
21             for(int j=0; j<data[i].length ; j++)
22                 data[i][j] = String.valueOf(value++);
23         }
24
25         defaultModel = new DefaultTableModel(data, name);
26         table=new JTable(defaultModel);
27         table.setPreferredScrollableViewportSize(new Dimension(400, 80));
28         JScrollPane s = new JScrollPane(table);
29
30         JPanel panel = new JPanel();
31         JButton b = new JButton("增加列");
32         panel.add(b);
33         b.addActionListener(this);

```

```
34         b = new JButton("增加行");
35         panel.add(b);
36         b.addActionListener(this);
37         b = new JButton("删除列");
38         panel.add(b);
39         b.addActionListener(this);
40         b = new JButton("删除行");
41         panel.add(b);
42         b.addActionListener(this);
43
44         Container contentPane = f.getContentPane();
45         contentPane.add(panel, BorderLayout.NORTH);
46         contentPane.add(s, BorderLayout.CENTER);
47
48         f.setTitle("AddRemoveCells");
49         f.pack();
50         f.setVisible(true);
51
52         f.addWindowListener(new WindowAdapter() {
53             public void windowClosing(WindowEvent e) {
54                 System.exit(0);
55             }
56         });
57     }
58
59     public void actionPerformed(ActionEvent e)
60     {
61         if(e.getActionCommand().equals("增加列"))
62             defaultModel.addColumn("增加列");
63         if(e.getActionCommand().equals("增加行"))
64             defaultModel.addRow(new Vector());
65         if(e.getActionCommand().equals("删除行"))
66         {
67             int columncount = defaultModel.getColumnCount()-1;
68             if(columncount >= 0)
69             {
70                 TableColumnModel columnModel = table.getColumnModel();
71                 TableColumn tableColumn = columnModel.getColumn(columncount);
72                 columnModel.removeColumn(tableColumn);
73                 defaultModel.setColumnCount(columncount);
74             }
75         }
76         if(e.getActionCommand().equals("删除列"))
77         {
78             int rowcount = defaultModel.getRowCount()-1;
79             if(rowcount >= 0)
80             {
81                 defaultModel.removeRow(rowcount);
82                 defaultModel.setRowCount(rowcount);
83             }
84         }
85         table.revalidate();
86     }
87
88     public static void main(String args[]) {
```

```

89         new AddRemoveCells();
90     }
91 }

```

说明:

- (1) 程序第 25 行, 利用 data 与 name 字符串数组产生 DefaultTableModel 对象。
- (2) 程序第 26 行, 设置表格模式为 DefaultTableModel。
- (3) 程序第 59~86 行, 处理用户按钮的操作。
- (4) 程序第 62 行, 当用户点选“增加列”按钮, 表格会增加一个标题为“增加列”的一列。
- (5) 程序第 64 行, 当用户点选“增加行”按钮, 表格会增加一空白行。
- (6) 程序第 67 行, getColumnCount()方法会返回列数, 若变量 columncount < 0 代表已经没有任何列了。
- (7) 程序第 70~74 行, 要删除列必须使用 TableColumnModel 界面定义的 removeColumn()方法。因此我们先由 JTable 类的 getColumnModel()方法取得 TableColumnModel 对象, 再由 TableColumnModel 的 getColumn()方法取得要删除列的 TableColumn。此 TableColumn 对象当作是 removeColumn()的参数。删除列完毕后必须重新设置列数, 也就是使用 DefaultTableModel 的 setColumnCount()方法来设置。
- (8) 程序第 78 行, getRowCount()方法会返回行数, 若变量 rowcount < 0 代表已经没有任何行了。
- (9) 程序第 81~82 行, 删除行比较简单, 只要用 DefaultTableModel 的 removeRow()方法即可。删除列完毕后必须重新设置列数, 也就是使用 DefaultTableModel 的 setRowCount()方法来设置。

程序运行结果如图 8-15 所示。



图 8-15

当新增 2 列, 删除 3 行后表格变成如图 8-16 所示。

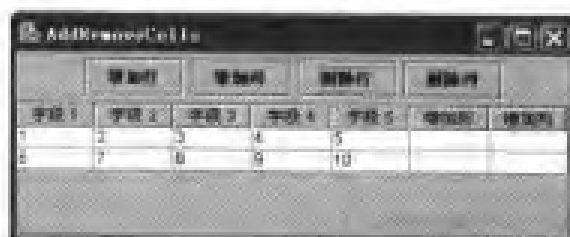


图 8-16

8-7 JTable 的事件处理

在前面几章，我们分别介绍过数种在不同组件上的事件处理。同样，在 JTable 上的事件处理大致均针对表格内容的异操作处理，包括字段内容改变、列数增加或减少、行数增加或减少、或是表格的结构改变等等，这些事件我们称为 TableModelEvent 事件。要处理 TableModelEvent 事件我们必须实现 TableModelListener 界面，此界面定义了一个方法，那就是 tableChanged()。为了处理这些事件的种种情况，在 AbstractTableModel 类中提供了下列方法来提示 TableModelListener 表格内容已经改动了，如下所示：

- fireTableCellUpdated(): 发出表格中的某一个字段已经更改的事件信息。
- fireTableChanged(): 发出表格已经改动的事件信息。
- fireTableDataChanged(): 发出表格中有字段已经更改的事件信息。
- fireTableRowsDeleted(): 发出表格中某几行已经删除的事件信息。
- fireTableRowsInserted(): 发出表格中已经新增某几行的事件信息。
- fireTableRowsUpdated(): 发出表格中某几行已经修改的事件信息。
- fireTableStructureChanged(): 发出表格结构已经改变的事件信息，这里指的结构改变可能包括表格的列数已经改变。

在知道了表格中可能发生的事件后，我们要如何去拦截这些事件的信息呢？在 AbstractTableModel 类中提供了一个注册 listener 的方法：addTableModelListener()。在加入 TableModelListener 之后，我们就可以依照不同的事件做不同的处理了。

先来看看下面这个例子吧，这个例子主要是更改本章的 ColumnModelTest.java 范例，再加上一些功能。在这个范例中，我们针对用户对表格所做的修改加以处理，如果修改的项目是数字，包括“语文”、“数学”字段，则我们直接将修改的值累加至“总分”字段，并检查其是否及格且在“及格”字段作修改；而当我们勾起“作弊”列的 Check Box 选项时，若原本总分应为及格者，则设置为不及格，且总分改成 119 分。

该范例的程序如下：

范例 TableEventHandle.java (文件位于随书光盘目录 exam\ch8\TableEventHandle.java)

```
1  import javax.swing.table.AbstractTableModel;
2  import javax.swing.event.*;
3  import javax.swing.table.*;
4  import javax.swing.*;
5  import java.awt.*;
6  import java.awt.event.*;
7  import java.util.*;
8
9  public class TableEventHandle implements TableModelListener
10 {
11     JTable table = null;
12     MyTable mt = null;
13     JLabel label = null; //显示修改字段位置
14
15     public TableEventHandle() {
16
```

```

17         JFrame f = new JFrame();
18         mt=new MyTable();
19         mt.addTableModelListener(this);
20
21         table=new JTable(mt);
22
23         JComboBox c = new JComboBox();
24         c.addItem("Taipei");
25         c.addItem("ChiaYi");
26         c.addItem("HsinChu");
27         table.getColumnModel().getColumn(1).setCellEditor(new DefaultCellEditor(c));
28
29         table.setPreferredScrollableViewportSize(new Dimension(550, 30));
30         JScrollPane s = new JScrollPane(table);
31
32         label = new JLabel("修改字段位置: ");
33         f.getContentPane().add(s, BorderLayout.CENTER);
34         f.getContentPane().add(label, BorderLayout.SOUTH);
35         f.setTitle("TableEventHandle");
36         f.pack();
37         f.setVisible(true);
38
39         f.addWindowListener(new WindowAdapter() {
40             public void windowClosing(WindowEvent e) {
41                 System.exit(0);
42             }
43         });
44     }
45
46     public void tableChanged(TableModelEvent e)
47     {
48         int row = e.getFirstRow();
49         int column = e.getColumn();
50         label.setText("修改字段位置: "+(row+1)+" 行 "+(column+1)+" 列");
51         boolean cheat=((Boolean) (mt.getValueAt(row,6))).booleanValue();
52         int grade1=((Integer) (mt.getValueAt(row,2))).intValue();
53         int grade2=((Integer) (mt.getValueAt(row,3))).intValue();
54         int total = grade1+grade2;
55         if(cheat)
56         {
57             if(total > 120)
58                 mt.mySetValueAt(new Integer(119),row,4);
59             else
60                 mt.mySetValueAt(new Integer(total),row,4);
61             mt.mySetValueAt(new Boolean(false),row,5);
62         }
63         else
64         {
65             if(total > 120)
66                 mt.mySetValueAt(new Boolean(true),row,5);
67             else
68                 mt.mySetValueAt(new Boolean(false),row,5);
69
70             mt.mySetValueAt(new Integer(total),row,4);

```

```

71         }
72         table.repaint();
73     }
74
75     public static void main(String args[]) {
76
77         new TableEventHandle();
78     }
79 }
80
81 class MyTable extends AbstractTableModel {
82
83     Object[][] p = {
84         {"阿呆", "Taipei", new Integer(66), new Integer(32), new Integer(98),
85          new Boolean(false), new Boolean(false)},
86         {"阿瓜", "ChiaYi", new Integer(85), new Integer(69), new Integer(154),
87          new Boolean(true), new Boolean(false)}};
88
89     String[] n = {"姓名", "居住地", "语文", "数学", "总分", "及格", "作弊"};
90
91     public int getColumnCount() {
92         return n.length;
93     }
94
95     public int getRowCount() {
96         return p.length;
97     }
98
99     public String getColumnName(int col) {
100         return n[col];
101     }
102
103     public Object getValueAt(int row, int col) {
104         return p[row][col];
105     }
106
107     public Class getColumnClass(int c) {
108         return getValueAt(0, c).getClass();
109     }
110
111     public boolean isCellEditable(int rowIndex, int columnIndex) {
112         return true;
113     }
114
115     public void setValueAt(Object value, int row, int col) {
116         p[row][col] = value;
117         fireTableCellUpdated(row, col);
118     }
119
120     public void mySetValueAt(Object value, int row, int col) {
121         p[row][col] = value;
122     }
123 }

```

说明:

- (1) 第 9 行, 由于我们要处理 `TableModelEvent` 事件, 因此我们必须实现 `TableModelListener` 界面。
- (2) 在程序第 19 行中, 我们利用 `AbstractTableModel` 类所提供的 `addTableModelListener()` 方法在表格中加入一个 `TableModelListener` listener, 这样就能检测出是否有 `TableModelEvent` 事件发生。
- (3) 程序第 46 行, `TableModelListener` 界面中只定义了一个方法, 那就是 `tableChanged()`, `tableChanged()` 方法用来处理 `TableModelEvent` 事件。在此范例中, 每当用户更改到表格内容时, 系统会自动调用在 `MyTable` 类中被我们所覆写(Override)的 `setValueAt()` 方法。我们在 `setValueAt()` 方法中调用 `AbstractTableModel` 抽象类所提供的 `fireTableCellUpdated()` 方法, 使得当用户更改表格内容后随即产生一个 `TableModelEvent` 事件。`MyTable` 类接收到此事件之后, 系统就会自动去运行 `tableChanged()` 方法, 这就是整个事件处理的过程。
- (4) 程序第 48~50 行, 我们分别读取更改字段的行列值, 并将此信息显示在 label 中。
- (5) 程序第 72 行, 使用 `repaint()` 方法可以让用户按完【Enter】键后, 表格内所有的内容马上跟着改动。
- (6) 程序第 120~122 行, 在此范例中, 由于我们改动一个字段内会牵引其他字段内容的改动, 因此系统会调用多次的 `setValueAt()` 方法, 并产生多次的 `TableModelEvent` 事件, 也将运行多次的 `tableChanged()` 方法。而事实上, 我们更改一个字段的内容时应只需要产生一次 `TableModelEvent` 事件, 因此只需要运行一次 `tableChanged()` 方法即可。为了避免产生过多的 `TableModelEvent` 事件, 我们重新定义了一个新的方法叫 `mySetValueAt()`, 用来更改表格字段的值而不会驱动表格更改的事件。如程序第 58 行。

现在表格会依照您所修改的字段而有所变化。在我们还没修改过表格前的状况如图 8-17 所示。

姓名	居住地	语文	数学	总分	及格	作弊
阿呆	Taipei	88	32	120	<input type="checkbox"/>	<input type="checkbox"/>
阿瓜	Chian	85	69	154	<input checked="" type="checkbox"/>	<input type="checkbox"/>

修改字段位置:

图 8-17

而在我们点选“作弊”的字段后, 状况如图 8-18 所示。

姓名	居住地	语文	数学	总分	及格	作弊
阿呆	Taipei	88	32	120	<input type="checkbox"/>	<input type="checkbox"/>
阿瓜	Chian	85	69	119	<input type="checkbox"/>	<input checked="" type="checkbox"/>

修改字段位置: 2 行 / 7 列

图 8-18

取消作弊字段后总分恢复原值。若将阿瓜的数学成绩改成 20 分，则表格自动计算出总分并判断出阿瓜成绩属于不及格，因此自动取消及格选项。结果如图 8-19 所示。

姓名	原住址	语文	数学	总分	及格	作弊
阿呆	Tapei	66	32	98	<input type="checkbox"/>	<input type="checkbox"/>
阿瓜	Cnaya	85	20	105	<input type="checkbox"/>	<input type="checkbox"/>

图 8-19

8-8 本章总结

本章中详细介绍了 `JTable` 类及其方法，而且也介绍了什么是 `TableModel`，以及 `TableModel` 与 `AbstractTableModel`、`DefaultTableModel` 之间的关系。并针对这三个 `Model` 做出了完整的介绍与范例解释，读者可由此内容得知在什么情况下用什么 `Model` 会比较方便。另外我们也介绍了 `SelectionModel` 与 `TableColumnModel` 的使用，这两个 `Model` 可以让我们对表格作出更具体的操作，例如增加或删除列或行、在表格中使用 `JComboBox`、只选取表格中某几个 `Cell` 数据等等。最后，我们也对表格的事件处理做了详细的介绍，利用表格的事件处理，您应该可以很轻松的使用 `JTable` 做出简易的电子表格。

8-9 本章习题

1. 请修改程序 `TableModel2.java`，使表格中的数据取自数据库，使多个级任老师可以看到他们所教的学生成绩。
2. 试述 `TableModel` 与 `AbstractTableModel`、`DefaultTableModel` 这三者之间的关系以及使用时机，并说明这三个 `Model` 的意义。
3. 继承 `AbstractTableModel` 时应自行实现哪些方法，请写出个范例进行解释。
4. 我们可以使用 `TableColumnModel` 让 `ComboBox` 置入表格的 `Cell` 中。同样的，请使用 `TableColumnModel` 让调色盘置于表格的 `Cell` 中，使表格的 `Cell` 具有选择颜色的功能（调色盘部分请参考后面章节）。
5. 利用表格的事件处理，试以 `JTable` 组件做出含有加法功能的简易电子表格。

9

文字输入组件的使用与介绍

大家一定有在网上填过抽奖问卷或在某个讨论区中发表过文章吧。不知道在您填这些数据时有没有发现除了我们以前介绍的互动组件外，还有一些需要让我们自行输入文字的部分，例如填姓名、生日、密码、建议或感想之类的。是不是很好奇这种组件该如何使用呢？如果您打算使您的网页更加有互动性，那么让我们快来看看本章的内容吧！

陈少一 张 强

9-1 认识 Swing 的文字输入组件

Swing 与文字输入有关的组件分别是 `JTextField`、`JPasswordField`、`JTextArea`、`JEditorPane` 与 `JTextPane`。`JTextField` 与 `JPasswordField` 为单行的文本编辑器；`JTextArea` 为多行的文本编辑器；`JEditorPane` 可显示多种文件格式；`JTextPane` 可设置文件各种样式。这些组件都继承了 `JTextComponent` 类，它们之间的关系如图 9-1 所示。

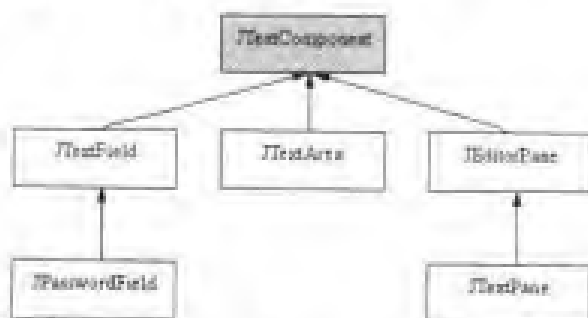


图 9-1

`JTextComponent` 提供了很多实用的方法，使处理输入组件更为方便，例如 `copy()`、`paste()`、`cut()`、`getText()`、`setText()` 等相当直观的方法；另外还有设置是否可编辑(`setEditable()`)、设置或取得文字选择区块 (`getSelectedText()`、`getSelectionEnd()`、`getSelectionStart()`、`setSelectionEnd()`、`setSelectionStart()`)、设置或取得光标位置 (`getCaretPosition()`、`setCaretPosition()`) 等等，这些相当常用的方法您都可以在 `JTextComponent` 类中找到。

Swing 的文字输入组件均以 `Document` 来当作数据模式，当输入组件的内容有所改变时，均是更改此 `Document` 的内容。因此您可以将同一个 `Document` 内容以不同的输入组件来显示，这就是 MVC 概念的一个基本应用（请参阅第 1 章）。`Document` 为一个 `Interface`，您可以实现此界面或利用 Java 提供的默认类来构造文字输入组件。图 9-2 是与 `Document` 相关的各种类关系图：

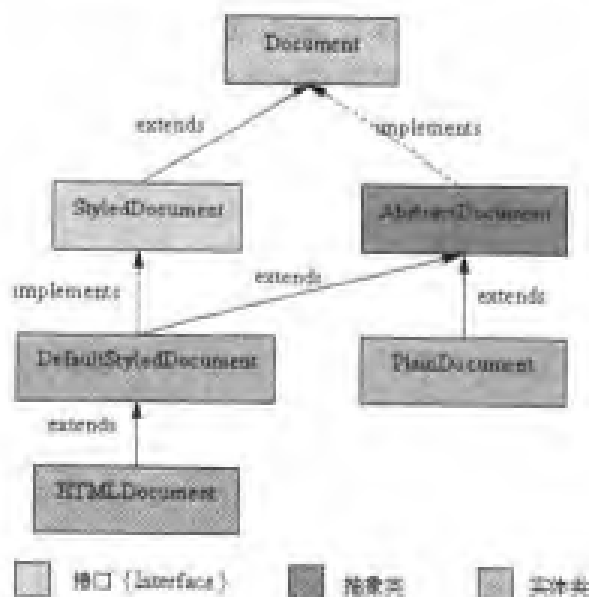


图 9-2

PlainDocument 是一个实体类, 实现了 AbstractDocument 与 Document 中的所有抽象方法, 您可以用此类直接构造出 JTextField、JPasswordField、与 JTextArea 组件; 相同的, 您可以使用 DefaultStyledDocument 构造出 JTextPane 组件, 这些关系我们均会在下面各节中提到。下面我们开始介绍各种文字输入组件的使用。

9-2 使用 JTextField 组件

JTextField 的类层次结构图:

```
java.lang.Object
  --java.awt.Component
    --java.awt.Container
      --javax.swing.JComponent
        --javax.swing.text.JTextComponent
          --javax.swing.JTextField
```

JTextField 继承了 JTextComponent 类, 因此它也可以使用 JTextComponent 抽象类里面许多好用的方法, 如 copy()、paste()、setText()、isEditable() 等等。我们可以在很多地方使用 JTextField, 例如在设计问卷时, 需要用户填入帐号或姓名等数据, 这时候 JTextField 就派上用场了。JTextField 是一个单行的输入组件, 那有没有多行的输入组件呢? 有的, 就是 JTextArea, 我们将会在后面几节中介绍。

在使用 JTextField 之前, 我们先看看 JTextField 有哪些构造函数可以使用, 如表 9-1 所示。

表 9-1

JTextField 的构造函数	
JTextField()	建立一个新的 JTextField
JTextField(Document doc, String text, int columns)	使用指定的文件存储模式建立一个新的 JTextField 并设置其初始字符串和字段长度
JTextField(int columns)	建立一个新的 JTextField 并设置其初始字段长度
JTextField(String text)	建立一个新的 JTextField 并设置其初始字符串
JTextField(String text, int columns)	建立一个新的 JTextField 并设置其初始字符串和字段长度

9-2-1 构造一般的 JTextField 组件

由上面 JTextField 所提供的构造函数可以看出, 我们可以很方便的构造出一个能够输入的 TextField, 并设置它的初始状态。以下为一个简单的 JTextField 的范例:

范例 JTextField1.java (文件位于随书光盘目录 exam\ch9\JTextField1.java)

```
1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4
```

```
5 public class JTextField1{
6
7     public static void main(String args[]) {
8
9         JFrame f = new JFrame("JTextField1");
10        Container contentPane = f.getContentPane();
11        contentPane.setLayout(new BorderLayout());
12
13        JPanel p1 = new JPanel();
14        p1.setLayout(new GridBagLayout());
15        GridBagConstraints gbc = new GridBagConstraints();
16        gbc.anchor = GridBagConstraints.WEST;
17        gbc.insets = new Insets(2,2,2,2);
18
19        p1.setBorder(BorderFactory.createTitledBorder("您的基本数据"));
20        JLabel l1 = new JLabel("姓名: ");
21        JLabel l2 = new JLabel("性别: ");
22        JLabel l3 = new JLabel("身高: ");
23        JLabel l4 = new JLabel("体重: ");
24        JTextField t1 = new JTextField();
25        JTextField t2 = new JTextField(2);
26        JTextField t3 = new JTextField("175cm");
27        JTextField t4 = new JTextField("50kg-太瘦了",10);
28
29        gbc.gridy=1;
30        gbc.gridx=0;
31        p1.add(l1,gbc);
32        gbc.gridx=1;
33        p1.add(t1,gbc);
34        gbc.gridy=2;
35        gbc.gridx=0;
36        p1.add(l2,gbc);
37        gbc.gridx=1;
38        p1.add(t2,gbc);
39        gbc.gridy=3;
40        gbc.gridx=0;
41        p1.add(l3,gbc);
42        gbc.gridx=1;
43        p1.add(t3,gbc);
44        gbc.gridy=4;
45        gbc.gridx=0;
46        p1.add(l4,gbc);
47        gbc.gridx=1;
48        p1.add(t4,gbc);
49
50        contentPane.add(p1);
51        f.pack();
52        f.show();
53        f.addWindowListener(new WindowAdapter() {
54            public void windowClosing(WindowEvent e) {
55                System.exit(0);
56            }
57        });
58    }
59 }
```

说明：

- (1) 程序第 11 行，我们将 contentPane 版面设为 BorderLayout。
- (2) 程序第 13 行，我们新建一个 JPanel，将之后的 4 个 JLabel 和 4 个 JTextField 放进此 JPanel 中。
- (3) 程序第 14 行，我们将此 JPanel 设为 GridBagLayout，并在程序第 15~16 行，分别设置 GridBagLayout 的位置和与各边界之间的间距。
- (4) 程序第 19 行，我们在此 JPanel 上加入边框 (Border)，并在边框上面加上“您的基本数据”的文字。
- (5) 程序第 24~27 行，利用 JTextField 构造函数的特性，建立 4 种 JTextField，并在程序第 29~48 行，依次加入 JPanel 中。

程序运行结果如图 9-3 所示。



图 9-3

我们可以看到四个不同长度的 JTextField，在第一个 JTextField 中，当我们在构造 JTextField 时并没有设置 JTextField 的字段长度，所以使用 JTextField 的字段长度默认值 0。第二个 JTextField 的字段长度设置为 2。第三个 JTextField 则是依字符串长度自动设置为最适当的字段长度。第四个 JTextField 的字段长度设置为 10，若是字段长度的设置小于字符串长度，则会有部分的字符串无法被显示出来。

另外，在使用到排列组件的 Layout Manager 时有一点要特别注意，不同的 Layout Manager 产生的排列效果可能不太一样。我们将上列程序中的 GridBagLayout 改成 GridLayout 的方式来比较这两种 Layout Manager 的差别。改后的程序如下：

范例 JTextField2.java (文件位于随书光盘目录 exam\ch9\JTextField2.java)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JTextField2{
6
7      public static void main(String args[]) {
8
9          JFrame f = new JFrame("JTextField2");
10         Container contentPane = f.getContentPane();
```

```
11     contentPane.setLayout(new BorderLayout());
12
13     JPanel p1 = new JPanel();
14     p1.setLayout(new GridLayout(4,2));
15
16     p1.setBorder(BorderFactory.createTitledBorder("您的基本数据"));
17     JLabel l1 = new JLabel("姓名: ");
18     JLabel l2 = new JLabel("性别: ");
19     JLabel l3 = new JLabel("身高: ");
20     JLabel l4 = new JLabel("体重: ");
21     JTextField t1 = new JTextField();
22     JTextField t2 = new JTextField(2);
23     JTextField t3 = new JTextField("175cm");
24     JTextField t4 = new JTextField("50kg-太瘦了",10);
25
26     p1.add(l1);
27     p1.add(t1);
28     p1.add(l2);
29     p1.add(t2);
30     p1.add(l3);
31     p1.add(t3);
32     p1.add(l4);
33     p1.add(t4);
34
35     contentPane.add(p1);
36     f.pack();
37     f.show();
38     f.addWindowListener(new WindowAdapter() {
39         public void windowClosing(WindowEvent e) {
40             System.exit(0);
41         }
42     });
43 }
44 }
```

◆ 说明:

- (1) 程序第 14 行, 我们将此 JPanel 设为 4 列 2 行的 GridLayout。
- (2) 程序第 26~33 行, 我们将 JLabel 和 JTextField 组件依次加入 JPanel 中。

◆ 程序运行结果如图 9-4 所示。



图 9-4

虽然我们在程序中有设置 JTextField 的字段长度值, 但是 GridLayout Manager 会忽略这些设置, 原因是 GridLayout 中的每一个格子都具有相同的长跟宽。因此在本例中, GridLayout 会选择

size 最大的组件作为每个格子的长与宽，即为 t4 这个组件。所以您可以看到上图的 JTextField 或是 JLabel 宽度都被拉大了。要解决这个问题除了使用不同的 Layout Manager 以外，您也可以将每一组 JLabel 与 JTextField 依次放入 4 个 FlowLayout 的 JPanel 上，也能解决这样的问题。

9-2-2 利用 Document 构造 JTextField

Document 是一个 Interface，主要的功能是定义一些方法，让我们在使用所有与 Text 相关的组件时，能够将输入的文字内容加以结构化或规格化。将文字内容结构化又是什么呢？举例来说，就好像一本书的内容，它的结构一定会有各个大章，在各章中又会分成许多小节，小节内会再有各个小重点等等，这样的树状组织结构就是结构化的一种。由于这个 Interface 定义了 10 几种方法，但是在这里我们所用到的只有少数几种方法，为了不让大家混淆，在此我们先列出这个 Interface 常被使用到的方法，有兴趣的读者可自行查阅 Java API 文件，如表 9-2 所示。

表 9-2

Document Interface 定义的方法（获取部分）	
void	addDocumentListener(DocumentListener listener) 增加 DocumentListener，使组件具有处理 DocumentEvent 功能
void	addUndoableEditListener(UndoableEditListener listener) 增加 UndoableEditListener，使组件具有处理 UndoableEditEvent 功能。当文件中的内容被修改时自动记忆可以被复原的内容
String	getText(int offset, int length) 取得 Document 中的文字内容
void	insertString(int offset, String str, AttributeSet a) 将字符串加入到 Text 组件的内容中
void	removeDocumentListener(DocumentListener listener) 移除 DocumentListener
void	removeUndoableEditListener(UndoableEditListener listener) 移除 UndoableEditListener，使复原功能失效

还记得我们一开始在介绍 JTextField 时所提到的构造函数吗？其中有一个 JTextField 的构造函数是这样写的：

```
JTextField(Document doc, String text, int columns)
```

因此我们必须实作 Document 所有的方法，才能利用 Document 构造出 JTextField。这样的做法有点麻烦，因为我们之前提到 Document 的方法有数十种，但是我们要用到的却只有其中几种，若是要将全部的方法实作那是相当费时的。大家还记得我们在前几章中介绍 JList 时，利用继承 AbstractListModel 的抽象类来构造 JList 吗？由于抽象类已经实作了许多界面的方法，所以当我们继承这个抽象类后便不需要再重新实作这些方法。同样的，Java 在这里也提供了一个 AbstractDocument 的抽象类来供我们使用。不过在这里我们并不是要使用 AbstractDocument，因为 Java 在这部分已经提供了一个实体类：PlainDocument。这个实体类继承了 AbstractDocument，也就是具备了所有 AbstractDocument 的方法，所以我们只要直接继承 PlainDocument 这个实体类就能利用 Document 来构造 JTextField。这种概念跟 JList 或是

JTable 的模式结构都是相同的。我们马上来看下面这个范例，在这个范例中我们限制了各个 JTextField 的输入长度，当输入长度超过我们的限制时程序会发出“哔”的一声来提示。

范例 JTextField3.java (文件位于随书光盘目录 exam\ch9\JTextField3.java)

在这个程序中，我们只将程序 JTextField1.java 的第 24~27 行做了修改，如下所示：

```
JTextField t1 = new JTextField(new JTextField3_FixedLengthDocument(10), "", 10);
JTextField t2 = new JTextField(new JTextField3_FixedLengthDocument(1), "", 2);
JTextField t3 = new JTextField(new JTextField3_FixedLengthDocument(5), "", 5);
JTextField t4 = new JTextField(new JTextField3_FixedLengthDocument(5), "", 5);
```

JTextField3_FixedLengthDocument 是我们自行设计的一个类，将在下面讨论。在这里我们将所有 JTextField 的构造函数换成使用 Document 的方式来构造，构造的参数意义分别是，可输入字符串的最大值 (Document 部分的参数)、默认字符串、字段长度。在第二个 JTextField 的设置中，虽然字段长度大于输入的最大值，可是在这里最多只能输入一个字。另外，重要的是这个 Document 该怎么样来设计呢？我们来看看下面的范例：

范例 JTextField3_FixedLengthDocument.java (文件位于随书光盘目录 exam\ch9\JTextField3_FixedLengthDocument.java)

```
1  import javax.swing.*;
2  import javax.swing.text.*;
3  import java.awt.Toolkit;
4
5  public class JTextField3_FixedLengthDocument extends PlainDocument{
6
7      private int maxLength;
8
9      public JTextField3_FixedLengthDocument(int maxLength){
10         this.maxLength = maxLength;
11     }
12
13     public void insertString(int offset,String str,AttributeSet att)
14     throws BadLocationException
15     {
16         if ( getLength() + str.length() > maxLength ){
17             Toolkit.getDefaultToolkit().beep();
18         }else{
19             super.insertString(offset,str,att);
20         }
21     }
22 }
```

说明：

- (1) 程序第 3 行，由于我们在程序中会用到“哔”的提示声，所以我们要将 java.awt.Toolkit Package 包含进来。
- (2) 程序第 5 行，继承 PlainDocument 类。
- (3) 程序第 9 行，在构造此类时传入 JTextField 可输入的最大长度。

- (4) 程序第 13 行, 覆写 insertString() 方法, 将其设置为当输入的长度大于最大长度时, 发出“哔”一声 (程序第 17 行), 若未超过最大长度则将字符串填入 JTextField 中 (程序第 19 行)。

程序运行结果如图 9-5 所示。



图 9-5

9-2-3 JTextField 的事件处理

在 JTextField 类中有 addActionListener() 方法, 可以检测用户是否在 JTextField 上按下【Enter】键, 就如同前面介绍 JButton 按下按钮时所产生的事件 (Event) 一样。我们马上来看下面这个范例:

范例 JTextField4.java (文件位于随书光盘目录 exam\ch9\JTextField4.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JTextField4{
6
7      public static void main(String args[]){
8
9          JFrame f = new JFrame("JTextField4");
10         Container contentPane = f.getContentPane();
11         contentPane.setLayout(new BorderLayout());
12
13         JPanel p1 = new JPanel();
14         p1.setLayout(new GridLayout(2,2));
15         p1.setBorder(BorderFactory.createTitledBorder("JTextField 事件处理范例"));
16         JLabel l1 = new JLabel("输入: ");
17         JLabel l2 = new JLabel("输入后, 按下 Enter ==>");
18         final JLabel l3 = new JLabel("");
19         final JTextField t1 = new JTextField();
20         t1.addActionListener(new ActionListener(){
21             public void actionPerformed(ActionEvent ev){
22                 l3.setText(t1.getText());
23             }
24         });
25         p1.add(l1);
26         p1.add(t1);
27         p1.add(l2);
28         p1.add(l3);
29     }

```

```

30     contentPane.add(p1);
31     f.pack();
32     f.show();
33     f.addWindowListener(new WindowAdapter() {
34         public void windowClosing(WindowEvent e) {
35             System.exit(0);
36         }
37     });
38 }
39 }

```

⊕ 说明：

- (1) 程序第 20 行，我们将此 `JTextField` 加入事件处理模式中。
- (2) 程序第 21 行，在这里我们采用 Inner Class 的匿名类写法来处理按下【Enter】键后所对应的事件，在这里是将 `JTextField` 中的字符串设置到一个 `JLabel` 上。

⊕ 程序运行结果如图 9-6 所示。



图 9-6

在 `JTextField` 中您可以使用 `setHorizontalAlignment()` 方法设置文字的排列方式，如居左、居中、或居右。可用的参数为：`JTextField.LEFT`、`JTextField.CENTER` 与 `JTextField.RIGHT`。

9-3 使用 JPasswordField 组件

`JPasswordField` 的类层次结构图：

```

java.lang.Object
--java.awt.Component
--java.awt.Container
--javax.swing.JComponent
--javax.swing.text.JTextComponent
--javax.swing.JTextField
--javax.swing.JPasswordField

```

一般我们在网络中填写登入密码时，密码栏都会显示“*”号代表用户输入的字符，这样可避免用户输入的密码信息被旁人看到。而 Swing 中的 `JPasswordField` 就可以提供这样的功能。`JPasswordField` 继承了 `JTextField` 类，因此它也可以使用 `JTextField` 类里面许多好用的方法，如 `addActionListener()`、`removeActionListener()`、`setHorizontalAlignment()` 等等。如同 `JTextField` 一样 `JPasswordField` 也是一个单行的输入组件，不同的是 `JPasswordField` 多了屏蔽 (Mask) 的功能，也就是说在 `JPasswordField` 中的字符都会以单一种的字符类型表现出来。

在使用 `JPasswordField` 之前，我们先来看一下如表 9-3 所示的 `JPasswordField` 的构造函数：

表 9-3

JPasswordField 的构造函数	
JPasswordField()	使用默认的文件存储模式建立一个新的 JPasswordField 并设置其初始字符串为空字符串和字段长度为 0
JPasswordField(Document doc, String text, int columns)	使用指定的文件存储模式建立一个新的 JPasswordField 并设置其初始字符串和字段长度
JPasswordField(int columns)	建立一个新的 JPasswordField 并设置其初始字段长度
JPasswordField(String text)	建立一个新的 JPasswordField 并设置其初始字符串
JPasswordField(String text, int columns)	建立一个新的 JPasswordField 并设置其初始字符串和字段长度

9-3-1 构造一般的 JPasswordField 组件

JPasswordField 的构造函数和 JTextField 的构造函数几乎一模一样, 唯一不同之处是在 JPasswordField 输入的字符时会以屏蔽字符的类型表示。我们来看下面这个范例:

范例 JPasswordField1.java (文件位于随书光盘目录 exam\ch9\JPasswordField1.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JPasswordField1{
6
7      public static void main(String args[]) {
8
9          JFrame f = new JFrame("JPasswordField1");
10         Container contentPane = f.getContentPane();
11         contentPane.setLayout(new BorderLayout());
12
13         JPanel p1 = new JPanel();
14         p1.setLayout(new GridBagLayout());
15         GridBagConstraints gbc = new GridBagConstraints();
16         gbc.anchor = GridBagConstraints.WEST;
17         gbc.insets = new Insets(2,2,2,2);
18
19         p1.setBorder(BorderFactory.createTitledBorder("您的基本数据"));
20         JLabel l1 = new JLabel("姓名: ");
21         JLabel l2 = new JLabel("性别: ");
22         JLabel l3 = new JLabel("身高: ");
23         JLabel l4 = new JLabel("体重: ");
24         JPasswordField t1 = new JPasswordField ();

```

```
25      JPasswordField t2 = new JPasswordField (2);
26      JPasswordField t3 = new JPasswordField ("175cm");
27      JPasswordField t4 = new JPasswordField ("50kg-太瘦了",10);
28
29      gbc.gridy=1;
30      gbc.gridx=0;
31      pl.add(l1,gbc);
32      gbc.gridx=1;
33      pl.add(t1,gbc);
34      gbc.gridy=2;
35      gbc.gridx=0;
36      pl.add(l2,gbc);
37      gbc.gridx=1;
38      pl.add(t2,gbc);
39      gbc.gridy=3;
40      gbc.gridx=0;
41      pl.add(l3,gbc);
42      gbc.gridx=1;
43      pl.add(t3,gbc);
44      gbc.gridy=4;
45      gbc.gridx=0;
46      pl.add(l4,gbc);
47      gbc.gridx=1;
48      pl.add(t4,gbc);
49
50      contentPane.add(pl);
51      f.pack();
52      f.show();
53      f.addWindowListener(new WindowAdapter() {
54          public void windowClosing(WindowEvent e) {
55              System.exit(0);
56          }
57      });
58  }
59 }
```

⊕ 说明:

- (1) 程序第 11 行, 我们将 `contentPane` 版面设为 `BorderLayout`。
- (2) 程序第 13 行, 我们产生一个 `JPanel`, 将之后的 4 个 `JLabel` 和 4 个 `JPasswordField` 放进此 `JPanel` 中。
- (3) 程序第 14 行, 我们将此 `JPanel` 设为 `GridBagLayout`, 并在程序第 15~16 行, 分别设置 `GridBagLayout` 的位置和与各边界之间的间距。
- (4) 程序第 18 行, 我们在此 `JPanel` 上加入边框 (`Border`), 并在边框上面加上“您的

基本数据”的文字。

(5) 程序第 24~27 行, 利用 JPasswordField 构造函数的特性, 建立 4 种 JPasswordField, 并在程序第 29~48 行, 依次加入 JPanel 中。

④ 程序运行结果如图 9-7 所示。



图 9-7

我们可以看到默认的字符串都以屏蔽字符“*”来表示, “*”字符是 JPasswordField 默认的屏蔽字符。我们可以利用 JPasswordField 提供的 setEchoChar()方法来改用其他字符作为屏蔽字符。我们来看下面这个范例:

范例 JPasswordField2.java (文件位于随书光盘目录 exam\ch9\JPasswordField2.java)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JPasswordField2{
6
7      public static void main(String args[])    {
8
9          JFrame f = new JFrame("JPasswordField2");
10         Container contentPane = f.getContentPane();
11         contentPane.setLayout(new BorderLayout());
12
13         JPanel p1 = new JPanel();
14         p1.setLayout(new GridLayout(4,2));
15
16         p1.setBorder(BorderFactory.createTitledBorder("您的基本数据"));
17         JLabel l1 = new JLabel("姓名: ");
18         JLabel l2 = new JLabel("性别: ");
19         JLabel l3 = new JLabel("身高: ");
20         JLabel l4 = new JLabel("体重: ");
21         JPasswordField t1 = new JPasswordField (10);
22         JPasswordField t2 = new JPasswordField (2);
23         JPasswordField t3 = new JPasswordField ("175cm");
24         JPasswordField t4 = new JPasswordField ("50kg-太瘦了",10);
25
26         t1.setEchoChar('#');
27         t2.setEchoChar('%');
28         t3.setEchoChar('&');
```

```
29         t4.setEchoChar('M');
30
31         p1.add(l1);
32         p1.add(t1);
33         p1.add(l2);
34         p1.add(t2);
35         p1.add(l3);
36         p1.add(t3);
37         p1.add(l4);
38         p1.add(t4);
39
40         contentPane.add(p1);
41         f.pack();
42         f.show();
43         f.addWindowListener(new WindowAdapter() {
44             public void windowClosing(WindowEvent e) {
45                 System.exit(0);
46             }
47         });
48     }
49 }
```

⊕ 说明:

- (1) 程序第 14 行, 我们将此 JPanel 设为 4 列 2 行的 GridLayout。由于 GridLayout 每一个格子都具有相同的长跟宽, 且会依据最大组件的大小来设置, 因此您可以看到下图中的每个组件都具有同样的大小。
- (2) 程序第 26~29 行, 我们将四个 JPasswordField 的屏蔽字符分别改为 “#”、“%”、“&”、“M”。
- (3) 程序第 31~38 行, 我们将 JLabel 和 JPasswordField 组件依次加入 JPanel 中。

⊕ 程序运行结果如图 9-8 所示。



图 9-8

9-3-2 利用 Document 构造 JPasswordField

我们在前面介绍过如何利用 Document 来构造 JTextField, 在构造 JPasswordField 中也是一样的。我们马上来看一个范例:

范例 JPasswordField3.java (文件位于随书光盘目录 exam\ch9\JPasswordField3.java)

在这个程序中, 我们只将程序 JPasswordField1.java 的第 24~27 行做了修改, 如下所示:

```
JPasswordField t1 = new JPasswordField (new
    JPasswordField3_OnlyNumberDocument (10), "", 10);
JPasswordField t2 = new JPasswordField (new
    JPasswordField3_OnlyNumberDocument (1), "", 2);
JPasswordField t3 = new JPasswordField (new
    JPasswordField3_OnlyNumberDocument (5), "", 5);
JPasswordField t4 = new JPasswordField (new
    JPasswordField3_OnlyNumberDocument (5), "", 5);
```

在这里我们设计了另一个 Document 来构造 JPasswordField, 这个 Document 的特性是只能输入数字字符 (0~9) 并且可以限制输入字符串的长度。我们来看看这个 Document 是怎么设计的吧:

范例 JPasswordField3_OnlyNumberDocument.java (文件位于随书光盘目录 exam\ch9\JPasswordField3_OnlyNumberDocument.java)

```
1  import javax.swing.*;
2  import javax.swing.text.*;
3  import java.awt.Toolkit;
4
5  public class JPasswordField3_OnlyNumberDocument extends PlainDocument{
6
7
8      private int maxLength;
9      int result;
10
11     public JPasswordField3_OnlyNumberDocument(int maxLength){
12         this.maxLength = maxLength;
13     }
14
15     public void insertString(int offset,String str,AttributeSet att)
16     throws BadLocationException
17     {
18         for(int i=0;i<=9;i++){
19             result = Integer.toString(i).compareTo(str);
20             if (result == 0){
21                 if ( getLength() + str.length() > maxLength ){
22                     Toolkit.getDefaultToolkit().beep();
23                 }else{
24                     super.insertString(offset,str,att);
25                 }
26             }
27         }
28     }
29 }
```


◆ 说明:

- (1) 程序第 3 行, 由于我们在程序中会用到“哔”的提示声, 所以我们要将 `java.awt.Toolkit` Package 包含进来。
- (2) 程序第 5~6 行, 继承 `PlainDocument` 类。
- (3) 程序第 11 行, 在构造此类时传入 `JPasswordField` 可输入的最大长度。
- (4) 程序第 15 行, 覆写 `insertString()` 方法, 先判断输入的字符是否为 0~9 的数字字符, 若是 (程序第 20 行) 则再判断当输入长度是否大于最大长度, 若是则发出“哔”一声 (程序第 22 行), 若未超过最大长度则将字符串填入 `JPasswordField` 中 (程序第 24 行)。

◆ 程序运行结果如图 9-9 所示。

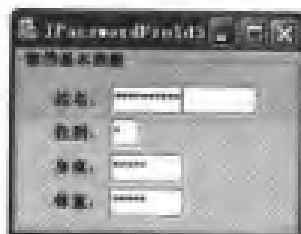


图 9-9

9-3-3 JPasswordField 的事件处理

由于 `JPasswordField` 继承了 `JTextField`, 因此在事件处理的部分也是依循 `JTextField` 的模式, 同样是使用 `addActionListener()` 方法来检测用户是否在 `JPasswordField` 上按下【Enter】键, 并且运行对应的事件 (Event)。我们马上来看下面这个范例:

范例 JPasswordField4.java (文件位于随书光盘目录 exam\ch9\JPasswordField4.java)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JPasswordField4{
6
7      public static void main(String args[]){
8
9          JFrame f = new JFrame("JPasswordField4");
10         Container contentPane = f.getContentPane();
11         contentPane.setLayout(new BorderLayout());
12
13         JPanel pl = new JPanel();
14         pl.setLayout(new GridLayout(2,2));
15         pl.setBorder(BorderFactory.createTitledBorder
16             ("JPasswordField 事件处理范例"));
17         JLabel ll = new JLabel("输入: ");
```

```

18         JLabel l2 = new JLabel("输入后, 按下 Enter ==>");
19         final JLabel l3 = new JLabel("");
20         final JPasswordField t1 = new JPasswordField();
21         t1.addActionListener(new ActionListener() {
22             public void actionPerformed(ActionEvent ev) {
23                 final char[] theStr = t1.getPassword();
24                 final String str = new String(theStr);
25                 l3.setText(str);
26             }
27         });
28         pl.add(l1);
29         pl.add(t1);
30         pl.add(l2);
31         pl.add(l3);
32
33         contentPane.add(pl);
34         f.pack();
35         f.show();
36         f.addWindowListener(new WindowAdapter() {
37             public void windowClosing(WindowEvent e) {
38                 System.exit(0);
39             }
40         });
41     }
42 }

```

说明：

- (1) 程序第 21 行, 我们将此 JPasswordField 加入事件处理模式中。
- (2) 程序第 22 行, 在这里我们采用 Inner Class 的匿名类写法来处理按下【Enter】键后所对应的事件, 在这里是将 JPasswordField 中的字符串设置到一个 JLabel 上(程序第 25 行)。
- (3) 程序第 23~24 行, 由于 JPasswordField 要使用 getPassword() 这个方法取得输入字符串的内容, 而其返回值是一个字符数组, 因此我们必须将字符数组转成字符串才能设置到 JLabel 中。

程序运行结果如图 9-10 所示。



图 9-10

9-4 使用 JTextArea 组件

JTextArea 的类层次结构图:

```

java.lang.Object
    --java.awt.Component

```

```
--java.awt.Container
--javax.swing.JComponent
--javax.swing.text.JTextComponent
--javax.swing.JTextArea
```

JTextArea 继承了 JTextComponent 类，因此它也可以使用 JTextComponent 抽象类里面许多好用的方法，如 copy()、paste()、setText()、isEditable() 等等。我们在前面有提到 JTextArea 是一个多行的输入组件，在这个组件中可以利用回车【Enter】键来做换行的操作。

在使用 JTextArea 之前，我们先看看 JTextArea 有哪些构造函数可以使用，如表 9-4 所示。

表 9-4

JTextArea 的构造函数	
JTextArea()	建立一个新的 JTextArea。
JTextArea(Document doc)	使用指定的文件存储模式建立一个新的 JTextArea
JTextArea(Document doc, String text, int row, int columns)	使用指定的文件存储模式建立一个新的 JTextArea 并设置其初始字符串和列、字段长度
JTextArea(int row, int columns)	建立一个新的 JTextArea 并设置其初始列、字段长度
JTextArea(String text)	建立一个新的 JTextArea 并设置其初始字符串
JTextArea(String text, int row, int columns)	建立一个新的 JTextArea 并设置其初始字符串和列、字段长度

9-4-1 构造的 JTextArea 组件

我们可以发现 JTextArea 的构造函数和 JTextField 及 JPasswordField 的构造函数是相当雷同的，而 JTextArea 多了一个字段的参数值是因为 JTextArea 是二维的输入组件，在构造时不仅要设置字段长度也要设置行数。我们马上来看一个范例：

范例 JTextArea1.java (文件位于随书光盘目录 exam\ch9\JTextArea1.java)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JTextArea1{
6
7      public static void main(String args[])    {
8
9          JFrame f = new JFrame("JTextArea1");
10         Container contentPane = f.getContentPane();
11         contentPane.setLayout(new BorderLayout());
12
13         JPanel p1 = new JPanel();
14         p1.setLayout(new GridBagLayout());
15         GridBagConstraints gbc = new GridBagConstraints();
16         gbc.anchor = GridBagConstraints.WEST;
```

```

17         gbc.insets = new Insets(2,2,2,2);
18
19         pl.setBorder(BorderFactory.createTitledBorder("构造一般的 JTextArea"));
20         JLabel l1 = new JLabel("一: ");
21         JLabel l2 = new JLabel("二: ");
22         JLabel l3 = new JLabel("三: ");
23         JLabel l4 = new JLabel("四: ");
24         JTextArea t1 = new JTextArea();
25         JTextArea t2 = new JTextArea(2,8);
26         JTextArea t3 = new JTextArea("JTextArea3");
27         JTextArea t4 = new JTextArea("JTextArea4",5,10);
28         t1.setText("JTextArea1");
29         t2.append("JTextArea2");
30         t4.setLineWrap(true);
31
32         gbc.gridy=1;
33         gbc.gridx=0;
34         pl.add(l1,gbc);
35         gbc.gridx=1;
36         pl.add(t1,gbc);
37         gbc.gridy=2;
38         gbc.gridx=0;
39         pl.add(l2,gbc);
40         gbc.gridx=1;
41         pl.add(t2,gbc);
42         gbc.gridy=3;
43         gbc.gridx=0;
44         pl.add(l3,gbc);
45         gbc.gridx=1;
46         pl.add(t3,gbc);
47         gbc.gridy=4;
48         gbc.gridx=0;
49         pl.add(l4,gbc);
50         gbc.gridx=1;
51         pl.add(t4,gbc);
52
53         contentPane.add(pl);
54         f.pack();
55         f.show();
56         f.addWindowListener(new WindowAdapter() {
57             public void windowClosing(WindowEvent e) {
58                 System.exit(0);
59             }
60         });
61     }
62 }
63

```

⊕ 说明:

- (1) 程序第 11 行, 我们将 contentPane 版面设为 BorderLayout。
- (2) 程序第 13 行, 我们产生一个 JPanel, 将 4 个 JLabel 和 4 个 JTextArea 放进此 JPanel 中。

- (3) 程序第 14 行，我们将此 JPanel 设为 GridBagLayout，并在程序第 16~17 行，分别设置 GridBagLayout 的位置和与各边界之间的间距。
- (4) 程序第 19 行，我们在此 JPanel 上加入边框 (Border)，并在边框上面加上“构造一般的 JTextArea”的文字。
- (5) 程序第 24~27 行，利用 JTextArea 构造函数的特性，建立 4 个 JTextArea，并在程序第 32~51 行，依次加入 JPanel 中。
- (6) 程序第 26 行，JTextArea 会取最适当的长度来存放字符串。
- (7) 程序第 28 行，以 setText() 方法将字符串填入第一个 JTextArea 中。程序第 29 行，以 append() 方法将字符串填入第二个 JTextArea 中。使用 setText() 方法会将原本在 JTextArea 的内容全部清除，再将设置的字符串填入 JTextArea 中。而 append() 方法则是会将设置的字符串接在原本 JTextArea 内容文字之后，不会将原有的内容删除。
- (8) 程序第 30 行，将第四个 JTextArea 的右边界固定为设置的字段长度。

⊕ 程序运行结果如图 9-11 所示。



图 9-11

在 JTextArea 中我们可以使用 setTabSize() 方法设置【Tab】键的跳离距离，或使 setFont() 方法设置文字字体。当我们输入的文字超过 JTextArea 的右边界及下边界时，会看不到下边输入的内容，那该怎么办呢？您可以使用 JScrollPane 使 JTextArea 具备滚动的能力，或是搭配 setLineWrap() 方法就能让文字自动换行。JTextArea 还提供了一个 setWrapStyleWord() 方法，可以让换行的时候不会造成断字的现象，这在 Word 或使用 Outlook 写信时都可以看到这个效果。例如我们在行尾输入“自动换行”四个字，但此行最多只能再容纳两个字，因此 JTextArea 会将这四个字都移到下一行，不会造成“自动”在上行，“换行”在下行的情形。这在处理英文输入上较为重要，因为 setWrapStyleWord() 是利用空白当作一个字输入的结束。我们来看下面的范例：

范例 JTextArea2.java (文件位于随书光盘目录 exam\ch9\JTextArea2.java)

```
1 import java.awt.*;  
2 import java.awt.event.*;
```

```

3      import javax.swing.*;
4
5      public class JTextArea2{
6
7          public static void main(String args[])    {
8
9              JFrame f = new JFrame("JTextArea2");
10             Container contentPane = f.getContentPane();
11             contentPane.setLayout(new BorderLayout());
12
13             JPanel p1 = new JPanel();
14             p1.setLayout(new GridLayout(1,1));
15             p1.setBorder(BorderFactory.createTitledBorder("构造 TextArea -使用
16             GridLayout,加 ScrollBar"));
17
18             JTextArea t1 = new JTextArea(5,25);
19             t1.setTabSize(10);
20             t1.setFont(new Font("标楷体", Font.BOLD, 16));
21             t1.setLineWrap(true);
22             t1.setWrapStyleWord(true);
23
24             p1.add(new JScrollPane(t1));
25
26             contentPane.add(p1);
27             f.pack();
28             f.show();
29             f.addWindowListener(new WindowAdapter() {
30                 public void windowClosing(WindowEvent e) {
31                     System.exit(0);
32                 }
33             });
34         }
35     }

```

◆ 说明:

- (1) 程序第 14 行, 我们将此 JPanel 设为 1 列 1 行的 GridLayout。
- (2) 程序第 19 行, 设置 JTextArea 里【Tab】键的跳离距离为 10 个默认字符长度, 若不设置则系统默认为跳离 8 个默认字符长度。
- (3) 程序第 20 行, 设置 JTextArea 内的输入字体。
- (4) 程序第 21 行, 激活自动换行功能。当输入长度大于 JTextArea 提供的宽度时, 会自动将超过的部分移到下一行, 但并不是真正换行。若您将 JTextArea 拉大, 您可以发现超过的部分还是在同一行上。
- (5) 程序第 22 行, 激活换行不断字的功能。
- (6) 程序第 24 行, 将 JTextArea 放入 JScrollPane 中, 这样就能利用滚动的效果看到输入超过 JTextArea 高度的文字了。

✎ 程序运行结果如图 9-12 所示。

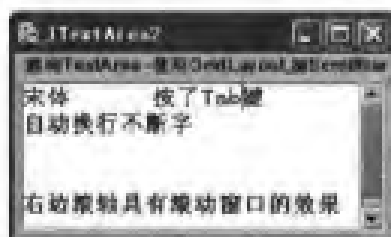


图 9-12

我们再举一个例子，使 JTextArea 具有 copy、paste、cut 的功能：

范例 JTextArea3.java (文件位于随书光盘目录 exam\ch9\JTextArea3.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class JTextArea3 implements ActionListener
6  {
7      JTextArea textarea = null;
8      JButton b1,b2,b3;
9
10     public JTextArea3()
11     {
12         JFrame f = new JFrame("JTextArea3");
13         Container contentPane = f.getContentPane();
14         contentPane.setLayout(new BorderLayout());
15
16         textarea = new JTextArea(10,15);
17         JScrollPane scrollPane = new JScrollPane(textarea);
18
19         JPanel panel = new JPanel();
20         panel.setLayout(new GridLayout(1,3));
21         b1 = new JButton("Copy");
22         b1.addActionListener(this);
23         b2 = new JButton("Paste");
24         b2.addActionListener(this);
25         b3 = new JButton("Cut");
26         b3.addActionListener(this);
27         panel.add(b1);
28         panel.add(b2);
29         panel.add(b3);
30
31         contentPane.add(scrollPane,BorderLayout.CENTER);
32         contentPane.add(panel,BorderLayout.SOUTH);
33
34         f.pack();
35         f.show();
36         f.addWindowListener(new WindowAdapter() {
37             public void windowClosing(WindowEvent e) {
38                 System.exit(0);
39             }
40         });
41     }
42 }
```

```
40     });  
41     }  
42  
43     public static void main(String args[])  
44     {  
45  
46         new JTextArea3();  
47     }  
48  
49     public void actionPerformed(ActionEvent e)  
50     {  
51         if (e.getSource() == b1)  
52         {  
53             textarea.copy();  
54         }  
55         if (e.getSource() == b2)  
56         {  
57             textarea.paste();  
58         }  
59         if (e.getSource() == b3)  
60         {  
61             textarea.cut();  
62         }  
63     }  
64 }
```

⊕ 说明:

- (1) 程序第 21~26 行, 建立 3 个按钮, 并处理按钮所产生的 `ActionEvent` 事件。
- (2) 程序第 49~63 行, 处理 `ActionEvent` 事件。要使 `JTextArea` 具有 `copy`、`paste`、`cut` 的功能相当简单, 只要个别调用 `JTextArea` 所提供的 `copy()`、`paste()`、`cut()` 方法即可。

⊕ 程序运行结果如图 9-13 所示。



图 9-13

另外, 使用 `Document` 来建立 `JTextArea` 的方式和步骤都和我们在 `JTextField` 及 `JPasswordField` 中介绍的一样, 唯一有差别的是在 `JTextArea` 中我们需要多设置一个行数的参数。因此我们在这里就不再多介绍如何利用 `Document` 来构造 `JTextArea` 了, 有兴趣的读者可以自己写写看。

9-4-2 JTextArea 的事件处理

由于 JTextArea 是一个二维的输入组件, 因此【Enter】键在 JTextArea 中代表的意义只是单纯的换行(归位)字符而不再是一个事件驱动的切入点。那么我们该如何来处理 JTextArea 的事件呢? 还记得我们在前面介绍过 Listener 的机制吗? 相同的, 我们一样需要使用 Listener 的机制来处理发生在 JTextArea 中的事件, 只是不再是以前提到的 ActionListener 了。在 JTextArea 中使用的 Listener 有两种, 一个是 UndoableEditListener, 另一个是 DocumentListener。UndoableEditListener Interface 是负责纪录 JTextArea 中所有操作发生的顺序并且可以运行还原上一步的功能。这个功能在目前的软件中应用相当广泛, 如文本编辑软件 Word 中的复原功能、画图板中的复原功能, 相信大家都有使用过。DocumentListener Interface 则是纪录发生在 JTextArea 中所有的事件(如键入字符、删除字符、剪切、粘贴)并将所有的事件以树状的层次结构组织起来; 也就是说当 JTextArea 中的内容有任何变动时, 就会产生 DocumentEvent, 此时必须使用 DocumentListener 界面中的方法来处理此事件。我们来看下面这个范例, 使 JTextArea 具有复原的功能:

范例 JTextArea4.java (文件位于随书光盘目录 exam\ch9\JTextArea4.java)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import javax.swing.undo.*;
5  import javax.swing.event.*;
6
7  public class JTextArea4 extends JFrame implements UndoableEditListener{
8
9      private UndoableEdit edit;
10     private JTextArea jta;
11     private JTextArea message;
12     private JMenuItem undoitem;
13     private JMenuItem redoitem;
14
15     public JTextArea4(){
16
17         super("JTextArea4");
18         jta = new JTextArea();
19         jta.getDocument().addUndoableEditListener(this);
20
21         message = new JTextArea();
22         message.setEditable(false);
23
24         JPanel p1 = new JPanel();
25         p1.setLayout(new GridLayout(1,1));
26         p1.setBorder(BorderFactory.createTitledBorder("Edit Area"));
27         p1.add(new JScrollPane(jta));
28
29         JPanel p2 = new JPanel();
30         p2.setLayout(new GridLayout(1,1));
31         p2.setBorder(BorderFactory.createTitledBorder("Message"));
32         p2.add(new JScrollPane(message));
33     }
```

```
34     getContentPane().setLayout(new GridLayout(2,1));
35     getContentPane().add(p1);
36     getContentPane().add(p2);
37
38     JMenuBar bar = new JMenuBar();
39     JMenu theMenu = new JMenu("Edit");
40     undoitem = new JMenuItem("Undo");
41     redoitem = new JMenuItem("Redo");
42     theMenu.add(undoitem);
43     theMenu.add(redoitem);
44     bar.add(theMenu);
45     updateMenuItem();
46
47     setJMenuBar(bar);
48     setSize(300,300);
49
50     undoitem.addActionListener(new ActionListener(){
51         public void actionPerformed(ActionEvent ev){
52             edit.undo();
53             updateMenuItem();
54             message.append("- Undo -\n");
55         }
56     });
57
58     redoitem.addActionListener(new ActionListener(){
59         public void actionPerformed(ActionEvent ev){
60             edit.redo();
61             updateMenuItem();
62             message.append("- Redo -\n");
63         }
64     });
65 }
66
67 public void undoableEditHappened(UndoableEditEvent ev){
68     StringBuffer buf = new StringBuffer(200);
69     edit = ev.getEdit();
70     buf.append("undoableEdit:");
71     buf.append(edit.getPresentationName());
72     buf.append("\n");
73     message.append(buf.toString());
74     updateMenuItem();
75 }
76
77 public void updateMenuItem(){
78
79     if (edit != null){
80         undoitem.setEnabled(edit.canUndo());
81         redoitem.setEnabled(edit.canRedo());
82         undoitem.setText(edit.getUndoPresentationName());
83         redoitem.setText(edit.getRedoPresentationName());
84     }else{
85         undoitem.setEnabled(false);
86         redoitem.setEnabled(false);
87         undoitem.setText("Undo");
88         redoitem.setText("Redo");
```

```
89         }
90     }
91
92     public static void main(String args[])    {
93
94         JFrame f = new JTextArea4();
95         f.addWindowListener(new WindowAdapter() {
96             public void windowClosing(WindowEvent e) {
97                 System.exit(0);
98             }
99         });
100        f.show();
101    }
102 }
```

◆ 说明:

- (1) 程序第 4~5 行, 由于会使用到复原和事件驱动功能, 因此需要将 `javax.swing.undo` 和 `javax.swing.event` 两个 Package 包含进来。
- (2) 程序第 94 行, 采用 Inner Class 的写作方法, 在这里新建一个名为 `JTextArea4` 的 `JFrame` 的类组件。
- (3) 程序第 7 行, `JTextArea4` 类继承 `JFrame` 类并实作 `UndoableEditListener` Interface。实作 `UndoableEditListener` Interface 就必须编写其中的 `undoableEditHappened()` 方法 (程序第 67~75 行)。
- (4) 程序第 19 行, 将 `JTextArea` 加入 `UndoableEditListener` 中。
- (5) 程序第 22 行, 利用 `setEditable()` 方法将另一个 `JTextArea` 设置为不可编辑。
- (6) 程序第 24~36 行, 分别将两个 `JTextArea` 通过 `JPanel` 放到 `JFrame` 中。
- (7) 程序第 38~45 行, 建立目录菜单并放置到 `JFrame` 中。
- (8) 程序第 50~64 行, 采用 Inner Class 方式, 分别构造目录菜单选项被点选后运行的操作。分别调用 `UndoableEdit` 的 `undo()` (程序第 52 行) 和 `redo()` (程序第 60 行) 方法来完成。
- (9) 程序第 69 行, 每当用户在 Text Area 中有所操作时, 就可以用 `getEdit()` 方法取得 `UndoableEdit` 对象, 此对象记录着刚才用户的操作, 因此可由此对象的 `undo()` 或 `redo()` 达到取消或复原的功能。
- (10) 程序第 45 行构造完目录菜单后、第 53 行运行完 `undo` 功能后、第 61 行运行完 `redo` 功能后、第 74 行当 `JTextArea` 发生每个事件后, 调用 `updateMenu()` 方法 (程序第 77~90 行) 来判断此时是否可以运行 `undo` 或 `redo` 的功能, 并且改变目录菜单的状态值。

◆ 程序运行结果如下:

1. 程序初始化后, 如图 9-14 (a) 所示。
2. 键入 2 个【Tab】键、key、1 个删除字符, 如图 9-14 (b) 所示。



图 9-14 (a)



图 9-14 (b)

3. 运行完 Undo 功能, 如图 9-15 (a) 所示。 4. 运行完 Redo 功能, 如图 9-15 (b) 所示。



图 9-15 (a)



图 9-15 (b)

我们在前面提到过【Enter】键在 JTextArea 中不再是事件驱动的切入点, 因此我们要利用 Listener 的机制来控制 JTextArea 的事件驱动。但是, 我们要怎么样知道在 JTextArea 中的数据内容呢? 这就要了解 JTextArea 的存储模式了, JTextArea 把输入区内的每一行当成一个独立的单元 (Element), 并依照 Document 内规划的树状结构来存储; 也就是说在 JTextArea 的第一行属于 Element 0、第二行属于 Element 1、第三行属于 Element 2 等等。不论我们在 JTextArea 内新增、插入或删除某一行, 系统会自动修改存储 Element 的树状结构, 不必我们手动去处理。下面我们来看看, Element 和 DocumentListener Interface 的用法。我们改写 JTextArea4.java 加入 DocumentListener, 将程序存储为 JTextArea5.java:

范例 JTextArea5.java (文件位于随书光盘目录 exam\ch9\JTextArea5.java)

注意: 部分程序代码请参考 JTextArea4.java, 在书中只列出重点部分。

```
1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4 import javax.swing.undo.*;
5 import javax.swing.event.*;
6 import javax.swing.text.*;
7
8 public class JTextArea5 extends JFrame implements
```

```
9      UndoableEditListener, DocumentListener{
10
11      private UndoableEdit edit;
12      private JTextArea jta;
13      private JTextArea message;
14      private JMenuItem undoitem;
15      private JMenuItem redoitem;
16
17      public JTextArea5(){
18
19      super("JTextArea5");
20      jta = new JTextArea();
21      jta.getDocument().addUndoableEditListener(this);
22      jta.getDocument().addDocumentListener(this);
23
24      ... 参照 JTextArea4.java ...
25      }
26
27      ... 参照 JTextArea4.java ...
28
29      public void showDE(DocumentEvent de){
30
31      StringBuffer debuf = new StringBuffer(100);
32      debuf.append(de.getType());
33      debuf.append(" Offset:");
34      debuf.append(de.getOffset());
35      debuf.append(" Length:");
36      debuf.append(de.getLength());
37
38      Element Eroot = jta.getDocument().getDefaultRootElement();
39      DocumentEvent.ElementChange Echange = de.getChange(Eroot);
40      if (Echange == null){
41          debuf.append(" (No Element Change)");
42      }else{
43          debuf.append(" Element Change: index ");
44          debuf.append(Echange.getIndex());
45      }
46      debuf.append("\n");
47      message.append(debuf.toString());
48      }
49
50      public void changedUpdate(DocumentEvent de){
51          showDE(de);
52      }
53
54      public void insertUpdate(DocumentEvent de){
55          showDE(de);
56      }
57
58      public void removeUpdate(DocumentEvent de){
59          showDE(de);
60      }
61
62      public static void main(String args[]) {
63
```

```

64     JFrame f = new JTextArea5();
65     f.addWindowListener(new WindowAdapter() {
66         public void windowClosing(WindowEvent e) {
67             System.exit(0);
68         }
69     });
70     f.show();
71 }
72 }

```

⊕ 说明:

- (1) 程序第 9 行, 除了 UndoableEditListener 外我们再实作 DocumentListener Interface。实作 DocumentListener Interface 就必须编写其三个方法, 分别是 changedUpdate() 方法 (程序第 50 行)、insertUpdate() 方法 (程序第 54 行)、removeUpdate() 方法 (程序第 58 行)。我们在这三个方法中都调用了 showDE() 方法来取得目前 JTextArea 内容的存储状况, 并将信息输出到信息窗口内。
- (2) 程序第 22 行, 将 JTextArea 加入 DocumentListener 中。
- (3) 程序第 38 行, 取得系统设置的 Document 之根节点。
- (4) 程序第 39 行, DocumentEvent.ElementChange 这个 Inner-Interface 定义了一个 Element 变化的集合, 因此我们利用它来得到 Element 节点的状态变化。
- (5) 程序第 44 行, 若是 Element 有变化时则取得变化根节点的 index 值。

⊕ 程序运行结果如图 9-16 所示。



图 9-16

在运行程序中, 当新增或删除一个 Element 时会有出现两次信息的情况是因为系统在处理 Element 存储位置的缘故。



注意

9-5 使用 JEditorPane 组件

JEditorPane 的类层次结构图:

```

java.lang.Object
--java.awt.Component

```

```
--java.awt.Container
--javax.swing.JComponent
--javax.swing.text.JTextComponent
--javax.swing.JEditorPane
```

JEditorPane 继承了 JTextComponent 类，因此它也可以使用 JTextComponent 抽象类里面的方法。JEditorPane 的最主要功能在于展现不同类型 (Types) 的文件格式内容。JEditorPane 支持的文件类型有三种：第一种是纯文本类型，其类型的表示法为 “text/plain”，这种类型的文件就是我们最常使用的 txt 文件，可以用记事本或 WordPad 等文本编辑软件来编辑。第二种是 RTF 类型，其表示法为 “text/rtf”，这种类型文件的特点是对文字内容做字体缩放、变形、上色等特殊效果，在编辑这种类型的文件时要使用 WordPad 或 Word 等支持 RTF 格式的文本编辑软件才行。第三类是 HTML 类型，也就是我们在网络上所浏览的网页类型，其表示法为 “text/html”，这类文件的特点相信大家都非常的清楚，除了在对字体效果的表现之外还具有在文件内加入图片、超级链接 (Hyper Link) 等相关功能。但是 JEditorPane 并不是一个全功能的 Web Browser，它仅能支持简单的 HTML 语法。JEditorPane 支持 HTML 类型的文件最主要的用途是用来制作在线辅助说明文件 (Online Help)，如表 9-5 所示。

表 9-5

JEditorPane 的构造函数	
JEditorPane()	建立一个新的 JEditorPane
JEditorPane(String url)	以详细的 URL 字符串为基础来建立一个 JEditorPane
JEditorPane(String type, String text)	建立一个被指定字符串 text 并指定初始化 JEditorPane 的类型
JEditorPane(URL initialPage)	以详细的 URL 字符串当作输入值来建立一个 JEditorPane

9-5-1 构造 JEditorPane 组件

我们从上面的列表中可以知道 JEditorPane 一共有四种构造方式，我们接下来一一为大家介绍。由于构造一个空的 JEditorPane 没有实质的意义，因此我们就将一个 HTML 文件放到构造完成的 JEditPane 中，请看下面的范例：

范例 JEditorPane1.java (文件位于随书光盘目录 exam\ch9\JEditorPane1.java)

```
1 import javax.swing.*;
2 import javax.swing.event.*;
3 import java.io.*;
4 import java.awt.event.*;
5
6 public class JEditorPanel{
7
8     public static void main(String[] args){
9
```

```
10         JEditorPane editPane = null;
11         try{
12             File file = new File ("docs/JEditorPane_1.html");
13             String str = file.getAbsolutePath();
14             str = "file:" + str;
15             editPane = new JEditorPane();
16             editPane.setPage(str);
17         }
18         catch(IOException ioe){
19             ioe.printStackTrace(System.err);
20             System.exit(0);
21         }
22         editPane.setEditable(false);
23
24         JFrame f = new JFrame("JEditorPanel");
25         f.setContentPane(new JScrollPane(editPane));
26         f.setSize(200, 250);
27         f.show();
28         f.addWindowListener(new WindowAdapter() {
29             public void windowClosing(WindowEvent e) {
30                 System.exit(0);
31             }
32         });
33     }
34 }
```

⊕ 说明：

- (1) 程序第 12 行，建立 JEditorPane_1.html 文件的 File 对象。
- (2) 程序第 13 行，取得文件所在位置的绝对路径。
- (3) 程序第 14 行，将绝对路径结合成一完整的输入字符串。
- (4) 程序第 15 行，构造一个空的 JEditorPane。
- (5) 程序第 16 行，利用 setPage()方法将字符串中路径所指的文件加载到 JEditorPane 中。在 setPage()方法中，输入的数据是 String 类型的字符串，其实这样的构造方式等同于利用 JEditorPane 的另一个构造函数 JEditorPane(String str)来构造。因此，如果我们将程序的第 15 和 16 行改写如下所示，会得到相同的效果。所以我们就不再对此种构造方式再多加说明了。

editPane = new JEditorPane(str);

- (6) 程序第 22 行，利用 setEditable()方法将 JEditorPane 设为不可编辑。请注意！这行是相当重要的！若是我们将这个方法设为 true，我们将会失去 HTML 文件本身的特性，如超级链接的功能等等。因此我们在下面 JEditorPane2 的例子时，一般都会将编辑的功能取消（设置为 false）。目前这个超链接功能并没有起作用，这部分将在 JEditorPane 的事件处理中作介绍。
- (7) 程序第 25 行，将 JEditorPane 加入滚动的功能。

◆ 程序运行结果如图 9-17 所示。

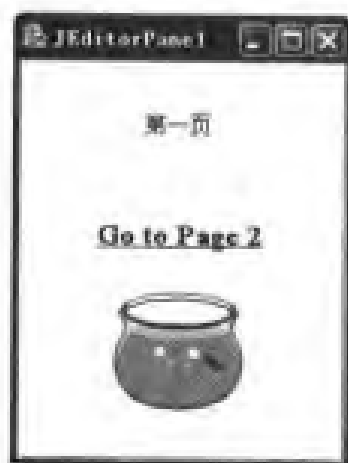


图 9-17

另外，如果我们输入不同格式的文件会出现什么样的情况呢？我们若是把程序第 12 行改成加载纯文本文件（如下），则结果会变为如图 9-18 所示。

```
File file = new File ("docs/JEditorPane_1.txt");
```



图 9-18

若是把程序第 12 行改成加载 RTF 格式的文本文件（如下），则结果会变为如图 9-19 所示。

```
File file = new File ("docs/JEditorPane_1.rtf");
```



图 9-19

我们在前面提到过 JEditorPane 支持三种类型的文件格式，在上面的例子里我们并没有看到设置文件格式的步骤，那是因为在上面的构造方法中系统会依照输入的文件名称来自动判

别文件类型。若是我们想要自己设置文件类型可利用 `setContentTypes()` 方法，或是直接在 `JEditorPane` 的构造函数中设置。如下面这个范例：

范例 JEditorPane2.java (文件位于随书光盘目录 exam\ch9\ JEditorPane2.java)

```
1  import javax.swing.*;
2  import javax.swing.event.*;
3  import java.awt.event.*;
4
5  public class JEditorPane2{
6
7      public static void main(String[] args){
8
9          String str = new String("This is a test.\nThis is Line 2!\nThis is
10             Line 3!");
11          JEditorPane editPane = new JEditorPane("text/plain",str);
12          editPane.setEditable(false);
13
14          JFrame f = new JFrame("JEditorPane2");
15          f.setContentPane(new JScrollPane(editPane));
16          f.setSize(200,250);
17          f.show();
18          f.addWindowListener(new WindowAdapter() {
19              public void windowClosing(WindowEvent e) {
20                  System.exit(0);
21              }
22          });
23      }
```

说明：

- (1) 程序第 9 行，设置文章内的默认文字内容。
- (2) 程序第 10 行，利用指定文件类型的构造函数来构造 `JEditorPane`。`JEditorPane` 支持的文件格式类型有“text/plain”、“text/rtf”、“text/html”。
- (3) 程序第 11 行，设置文件为不可编辑。

程序运行结果如图 9-20 所示。

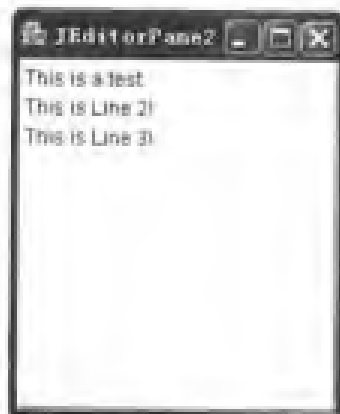


图 9-20

还有一种构造 `JEditorPane` 的方式是以 `URL` 类（指定超链接的位置）当作 `JEditorPane` 的参数来构造。但是要注意的地方是使用这种方式来构造时，计算机需要连接上局域网或网际网络，不然程序会找不到 `URL` 指定的位置而产生 `Exception` 使得程序无法运作。我们马上来看一个范例吧！

范例 `JEditorPane3.java` (文件位于随书光盘目录 `exam\ch9\JEditorPane3.java`)

```
1   import javax.swing.*;
2   import javax.swing.event.*;
3   import java.awt.event.*;
4   import java.net.*;
5   import java.io.*;
6
7   public class JEditorPane3{
8
9       public static void main(String[] args){
10
11           JEditorPane editPane = null;
12           try{
13               URL address = new URL("http://www.chinamor.cn.net");
14               editPane = new JEditorPane(address);
15           }catch(MalformedURLException e){
16               System.out.println("Malformed URL: " + e);
17           }catch(IOException e){
18               System.out.println("IOException: " + e);
19           }
20           editPane.setEditable(false);
21
22           JFrame f = new JFrame("JEditorPane3");
23           f.setContentPane(new JScrollPane(editPane));
24           f.setSize(200,250);
25           f.show();
26           f.addWindowListener(new WindowAdapter() {
27               public void windowClosing(WindowEvent e) {
28                   System.exit(0);
29               }
30           });
31       }
32   }
```

⊕ 说明：

- (1) 程序第 4 行，由于 `URL` 类属于 `java.net` 下的类，因此要将 `java.net` 的 `Package` 包含进来。
- (2) 程序第 13 行，构造 `URL` 的 `Instance` 并指定 `URL` 所指的超链接位置，在这里我们指定中华铁路网的首页 (`http://www.chinamor.cn.net`) 为默认位置。
- (3) 程序第 20 行，将 `JEditorPane` 设置为不可编辑。

◆ 程序运行结果如图 9-21 所示。



图 9-21

若是当计算机没有连上网络而运行程序时，则会出现 `IOException`，如图 9-22 所示。



图 9-22

9-5-2 JEditorPane 的事件处理

在 `JEditorPane` 文件中最常用到事件处理的部分就是 `HTML` 文件，那是因为 `HTML` 文件本身具有超级链接的功能来做文章链接的用途。大家还记得这一节的第一个范例吗？我们不是加载了一份 `HTML` 文件到 `JEditorPane` 中吗，虽然画面上都确实将超级链接和图片等信息展现出来了，可是你有没有发现当你想要點選超级链接时却没有反应呢？那是因为我们并没有在 `JEditorPane` 中加入相关事件处理机制的缘故。我们改写 `JEditorPanel.java` 加入相关的事件处理机制，使 `JEditorPane` 具有正常的超级链接功能。如下面这个范例所示：

范例 `JEditorPane4.java` (文件位于随书光盘目录 `exam\ch9\JEditorPane4.java`)

```
1 import javax.swing.*;
2 import javax.swing.event.*;
```

```
3    import java.io.*;
4    import java.awt.event.*;
5
6    public class JEditorPanel{
7
8        public static void main(String[] args){
9
10           JEditorPane editPane = null;
11           try{
12               File file = new File ("docs/JEditorPane_1.html");
13               String str = file.getAbsolutePath();
14               str = "file:"+str;
15               editPane = new JEditorPane();
16               editPane.setPage(str);
17           }
18           catch(IOException ioe){
19               ioe.printStackTrace(System.err);
20               System.exit(0);
21           }
22           editPane.setEditable(false);
23
24           final JEditorPane thePane = editPane;
25           editPane.addHyperlinkListener(new HyperlinkListener(){
26               public void hyperlinkUpdate(HyperlinkEvent hle){
27                   try{
28                       if (hle.getEventType() ==HyperlinkEvent.EventType.ACTIVATED)
29                           thePane.setPage(hle.getURL());
30                   }
31                   catch(IOException ioe){
32                       ioe.printStackTrace(System.err);
33                   }
34               }
35           });
36
37           JFrame f = new JFrame("JEditorPanel");
38           f.setContentPane(new JScrollPane(editPane));
39           f.setSize(200,250);
40           f.show();
41           f.addWindowListener(new WindowAdapter() {
42               public void windowClosing(WindowEvent e) {
43                   System.exit(0);
44               }
45           });
46       }
47   }
```

◆ 说明:

- (1) 程序第 24~34 行, 是处理超级链接事件的部分。
- (2) 程序第 25 行, 利用 addHyperlinkListener()方法将 JEditorPane 加入对超级链接事件的处理机制。

- (3) 程序第 25 行, 采用 Inner Class 的方式编写触发超级链接事件时的对应操作类。
- (4) 程序第 26 行, 覆写 `hyperlinkUpdate()` 方法, 当超级链接事件触发时会进入这个区段运行。
- (5) 程序第 28 行, 判断是否为超级链接的链接操作。若操作为真, 则将新的 HTML 文件放到 `JEditorPane` 中 (程序第 29 行)。

✎ 程序运行结果如下:

1. 程序开始运行后, 如图 9-23 (a) 所示。
2. 点选超级链接 `Go to Page 2` 后, 如图 9-23 (b) 所示。

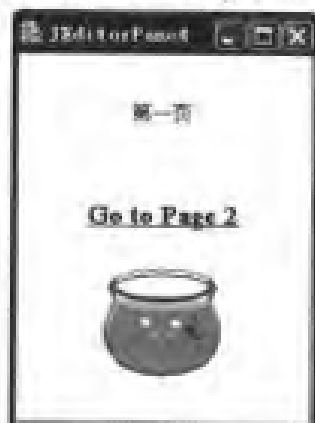


图 9-23 (a)

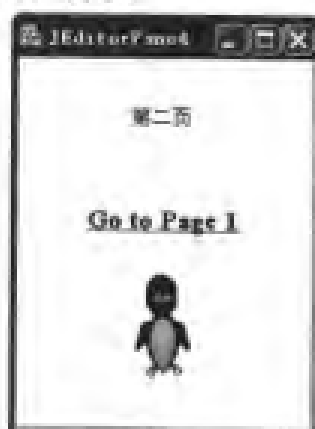


图 9-23 (b)

另外, 若我们是在纯文本模式 (Plain Text) 或 RTF 模式 (RTF Text) 下需要事件处理模式又该怎么办呢? 您还记得我们在 `JTextArea` 中是如何加入事件处理模式的吗? 没错! 在 `JEditorPane` 中也采用了相同的做法, 也就是利用 `DocumentListener Interface` 的机制来处理, 由于做法类似, 因此我们就不在这里重复说明了。

9-6 使用 `JTextPane` 组件

`JTextPane` 的类层次结构图:

```
java.lang.Object
--java.awt.Component
--java.awt.Container
--javax.swing.JComponent
--javax.swing.text.JTextComponent
--javax.swing.JEditorPane
--javax.swing.JTextPane
```

我们在前面有介绍过 `JTextArea` 类, 虽然 `JTextArea` 在某些功能上已经能够满足我们的需求, 但是当我们想再加入更多的变化时 (如文字加入色彩、插入图片……) 就会发现 `JTextArea` 类根本无法做到。要做到这些功能, 我们必须使用 `JEditorPane` 的子类: `JTextPane`。 `JTextPane` 提供了许多对文字处理的功能, 如改变颜色、缩放字体、文字风格、加入图片等。我们先来看看 `JTextPane` 的构造函数, 如表 9-6 所示。

表 9-6

JTextPane 的构造函数

JTextPane()
建立一个新的 JTextPane
JTextPane(StyledDocument doc)
以指定的文件模式建立一个新的 JTextPane

9-6-1 JTextPane 的特性

相信大家都有用 Word 写过报告或文章,那么你一定会知道我们在 Word 中可以对文章中的文字做很多的变化,这些变化都是属于文字的“属性”变化。由于在 JTextPane 中产生的效果几乎都是由属性的变化而来,所以改变属性的类组件在 JTextPane 中是少不了的。因此在介绍如何构造 JTextPane 之前,我们要先介绍两个在 JTextPane 中常用到的属性类: SimpleAttributeSet 和 StyleConstant。属性的变化原本是利用 AttributeSet Interface 来处理的,但是这个 Interface 中包含了太多的方法,若是我们直接实作 AttributeSet Interface 那就需要实作此 Interface 里所有的方法,这对于编写程序来说并不是一个很理想的做法;而 Java 另外提供了 SimpleAttributeSet 类,实作了 AttributeSet Interface。因此,只要我们直接使用 SimpleAttributeSet 类就能具备 AttributeSet Interface 的所有功能,而不用一个个的编写 AttributeSet 中的方法。另外 StyleConstant 类则是提供给 AttributeSet 类许多常用的属性键值(Attribute Key)和方法来设置或取得 JTextPane 内容的状态。在 StyleConstant 类中包含了许多的常用的属性设置,包括本文与边界空白区段设置、文字字体/大小/类型设置、背景颜色设置等。利用这两个类来辅助设计 JTextPane 就能使 JTextPane 具有更丰富的内容变化。

JTextPane 是专为文字和版面处理设计的组件。JTextPane 对可输入区域内容(就是可以打字区域)的设计概念是一个类似 Word 的设计概念,也就是说在 JTextPane 中的文字结构是有“段落”概念的。“段落”的概念就是以【Enter】键为每一段落的分界点,每按一次【Enter】键就增加一个段落。记得我们在 JTextArea 中提过的 Element 存储模式吗?在 JTextPane 中也采用了相同的做法,但是差别在于规划存储的方式不同。在 JTextArea 中并没有分段落,只是单纯的以【Enter】键当作存储成两个 Element 的分界。而在 JTextPane 中则是以整个编辑区域为根节点(Root),每个段落为枝节点(Branch Element),每个字符为叶节点(Leaf Element)来存储文件。也因为 JTextPane 是采用这样的方式来存储数据,因此在 JTextPane 中也可以像 Word 文件一样将各个段落设置成不同的属性,如第一段为斜体字、字体大小为 14 号字、粗体字,第二段为斜体字、字体颜色为蓝色、向左边界缩排 2 厘米等;另外,我们还可以设置 JTextPane 编辑区内输入的文字与各个边界间的距离。由这些功能看来,对于一个 Text Component 来说 JTextPane 是一个具有很多实用功能的组件。

9-6-2 构造 JTextPane 组件

在了解 JTextPane 的各项特性之后,我们现在马上来看 JTextPane 可以呈现什么样的效果。在下面这个例子中我们将对 JTextPane 区域内的文字设置颜色、粗斜体、与底线等相关属性:

范例 JTextPane1.java (文件位于随书光盘目录 exam\ch9\JTextPanc1.java)

```

1  import javax.swing.*;
2  import javax.swing.text.*;
3  import java.awt.event.*;
4  import java.awt.*;
5
6  public class JTextPanel{
7
8      private JTextPane textPane;
9
10     public JTextPanel()
11     {
12         textPane = new JTextPane();
13         textPane.setBackground(Color.black);
14         textPane.setEditable(false);
15     }
16
17     public void setYellow_Bold_20(String str)
18     {
19         SimpleAttributeSet attrset = new SimpleAttributeSet();
20         StyleConstants.setForeground(attrset, Color.yellow);
21         StyleConstants.setBold(attrset, true);
22         StyleConstants.setFontSize(attrset, 20);
23         insert(str, attrset);
24     }
25
26     public void setBlue_Italic_Bold_22(String str)
27     {
28         SimpleAttributeSet attrset = new SimpleAttributeSet();
29         StyleConstants.setForeground(attrset, Color.blue);
30         StyleConstants.setItalic(attrset, true);
31         StyleConstants.setBold(attrset, true);
32         StyleConstants.setFontSize(attrset, 22);
33         insert(str, attrset);
34     }
35
36     public void setRed_UnderLine_Italic_24(String str)
37     {
38         SimpleAttributeSet attrset = new SimpleAttributeSet();
39         StyleConstants.setForeground(attrset, Color.red);
40         StyleConstants.setUnderline(attrset, true);
41         StyleConstants.setItalic(attrset, true);
42         StyleConstants.setFontSize(attrset, 24);
43         insert(str, attrset);
44     }
45
46     public void insert(String str, AttributeSet attrset)
47     {
48         Document docs = textPane.getDocument();
49         str = str + "\n";
50         try{
51             docs.insertString(docs.getLength(), str, attrset);
52         }catch(BadLocationException ble){
53             System.out.println("BadLocationException: "+ble);

```



```
54     }
55 }
56
57 public Component getComponent(){
58     return textPane;
59 }
60
61 public static void main(String[] args){
62
63     JTextPanel pane = new JTextPanel();
64     pane.setYellow_Bold_20("This is Line 1, yellow, Bold, Size 20");
65     pane.setBlue_Italic_Bold_22("This is Line 2, blue, Italic, Bold,
66     Size 22");
67     pane.setRed_UnderLine_Italic_24("This is Line 3, red, UnderLine,
68     Italic, Size 24");
69
70     JFrame f = new JFrame("JTextPanel");
71     f.getContentPane().add(pane.getComponent());
72     f.setSize(450,180);
73     f.show();
74     f.addWindowListener(new WindowAdapter() {
75         public void windowClosing(WindowEvent e) {
76             System.exit(0);
77         }
78     });
79 }
```

⊕ 说明:

- (1) 程序第 63 行, 新建一个 `JTextPanel` 的对象 `pane`。
- (2) 程序第 64~66 行, 调用 `pane` 中的 3 个方法, 分别在 `JTextPane` 中显示三行不同字体变化的效果。
- (3) 程序第 12 行, 在 `JTextPanel` 的构造函数中构造 `JTextPane` 组件。
- (4) 程序第 13 行, 利用 `setBackground()` 方法将 `JTextPane` 组件的背景设为黑色。
- (5) 程序第 14 行, 将 `JTextPane` 设置为不可编辑。
- (6) 程序第 17~24 行, 为 `setYellow_Bold_20()` 方法, 将传入的字符串设置为黄色、粗体字、字体大小为 20 号。在程序第 19 行新建一个新的 `SimpleAttributeSet` 对象作为 `StyleConstants` 类中方法的参数值。程序第 20 行, 利用 `StyleConstants.setForeground()` 方法将字体颜色设为黄色。程序第 21 行, 利用 `StyleConstants.setBold()` 方法将字体类型设为粗体字。程序第 22 行, 利用 `StyleConstants.setFontSize()` 方法将字体大小设置为 20 号字。程序第 23 行, 调用 `insert()` 方法将字符串插入到 `JTextPane` 中。
- (7) 程序第 46~55 行, 为 `insert()` 方法, 这个方法最主要的用途是将字符串插入到 `JTextPane` 中。程序第 48 行, 利用 `getDocument()` 方法取得 `JTextPane` 的 `Document` `Instance`。程序第 49 行, 将输入的字符串加入换行字符。程序第 51 行, 利用 `insertString()` 方法将字符串插入到 `JTextPane` 中。

- (8) 程序第 26~34 行, 为 `setBlue_Italic_Bold_22()` 方法, 将传入的字符串设置为蓝色、斜和粗体字、字体大小为 22 号。其中程序第 30 行, `StyleConstants.setItalic()` 方法将字体类型设为斜体字。
- (9) 程序第 36~44 行, 为 `setRed_UnderLine_Italic_24()` 方法, 将传入的字符串设置为红色、字体加下划线、字体大小为 24 号。其中程序第 40 行, 使用 `StyleConstants.setUnderline()` 方法将字体类型设为字体加下划线。
- (10) 程序第 69 行, 利用 `getComponent()` 方法返回 `JTextPane` 的 `Component` 并加入到 `JFrame` 中。

⊕ 程序运行结果如图 9-24 所示。



图 9-24

若您想在 `JTextPane` 上置入图形或其他组件 (如表格或按钮), 您可以分别使用 `JTextPane` 所提供的 `setIcon()` 与 `insertComponent()` 方法来达到这个效果。

至于另外一种 `JTextPane` 的构造方式和 `JTextArea` 一样, 差别在于 `JTextArea` 是采用 `Document Interface` 而 `JTextPane` 是采用 `StyledDocument`。`StyledDocument` 继承自 `Document` 因此具有 `Document Interface` 的所有特性再加上一些改变属性设置的方法, 因此在这里我们就不对这种构造方式多做介绍了。

我们在前面提到过 `JTextPane` 是专为文字和版面处理设计的组件, 因此当我们在使用 `JTextPane` 时大多是着重在文字内容的表现和编排上, 而较少使用到事件的处理, 若是需要使用到事件处理的机制则和 `JTextArea` 及 `JEditorPane` 的做法一样, 都是使用 `DocumentListener` 来处理即可。

9-7 本章总结

在此章中, 我们分别介绍了 `JTextField`、`JPasswordField`、`JTextArea`、`JEditorPane` 和 `JTextPane` 的特性、使用方法及事件处理的模式。一般在制作 Web 上的文件时比较常用的是 `JTextField`、`JPasswordField` 和 `JTextArea`, 这些也就是我们在网络上常会看到及使用到的界面工具。因此, 仔细学习使用这些组件在制作网页互动时会有不小的帮助。另外 `JEditorPane` 大多应用在加载不同格式类型的文件上。`JTextPane` 则是提供文字内容的变化与排版的设置, 您可以利用 `JEditorPane` 制作具有简易排版功能的文本编辑器。这两类组件多半是在编写 `Application` 时会使用到, 而前三类组件则是在编写 `Application` 和 `Applet` 时上都经常用到。只要读者能活用这些组件, 相信在您编写程序时一定会更加的简单和方便。

1. 试利用 Document 来构造一个 JTextArea。
2. 试利用 UndoableEditListener 写出可恢复多步骤的 JTextArea。
3. 试利用 Document 的事件处理机制来编写一个具备“恢复上一步操作”的 JEditorPane（在纯文本格式或 RTF 格式下）。
4. 试在 JTextPane 中加入表格与按钮，并设计按钮的事件处理模式（ActionListener）。
5. 试利用 JTextPane 的功能编写一个如图 9-25 所示的排版及文字变化效果的程序。

Java Swing

Chapter 1. Overview

Chapter 2. Introduction

Chapter 3. Swing TextComponent

Section 1. Labels

Section 2. Text Fields

Chapter 4. Basic Controls

Section 1. Checkboxes

Chapter 5. Menus

图 9-25

10

树(Tree)的使用与介绍

相信读者一定使用过文件管理器,在文件管理器中各个数据以层次式结构显示出来,就如同阶梯一般,一层包着一层,这样的做法不仅可以让用户清楚地了解各节点之间的关系,也使得用户在找寻相关的数据时更为方便。这样的显示方式,Java 的 JTree 可以让您轻松地做到。Tree 是 Swing 才有的组件,在本章中我们会先介绍如何以最简单的方式来构造 JTree,之后以较复杂的方式构造 JTree,使它具有较强大的功能,例如找到节点的父节点或子节点、设置节点图案、改变节点外观等。最后我们介绍 JTree 的事件处理模式,让用户对 JTree 的操作能得心应手。

10-1 使用 JTree 组件

JTree 的类层次结构图:

```
java.lang.Object
--java.awt.Component
--java.awt.Container
--javax.swing.JComponent
-- javax.swing.JTree
```

相信读者一定使用过文件管理器, 在文件管理器中各个数据以层次结构显示出来, 就如同阶梯一般, 一层包着一层, 这样的做法不仅可以让用户清楚地了解各节点之间的关系, 也使得用户在找寻相关的数据时更为方便。

一般树的结构, 我们称最上层的点为根节点 (Root Node), 在根节点下面的节点我们称为子节点 (Child Node), 子节点下面还可以有子节点, 因此就会形成所谓父节点与子节点的关系。每个节点可以有零个到多个的子节点, 当一个节点没有任何的子节点时, 我们就称这个节点为树叶节点 (Leaf Node), 反之我们称为树枝节点 (Internal Node)。如图 10-1 所示的关系:

节点 A 为根节点, 因此节点 B、C、D、E、F 为 A 的子节点。节点 C 是节点 D、E、F 的父节点, 节点 D、E、F 是节点 C 的子节点。节点 B、D、E、F 为树叶节点, 节点 A、C 为树枝节点。

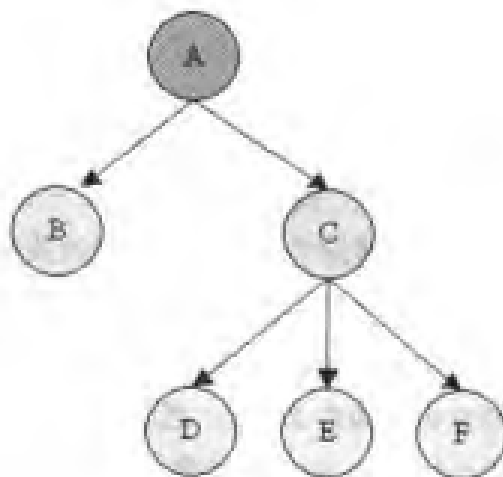


图 10-1

要如何建立一棵树呢? 我们先来看 JTree 的构造函数, 如表 10-1 所示。

表 10-1

JTree 的构造函数

JTree()

建立一棵系统默认的树

JTree(Hashtable value)

利用 Hashtable 建立树, 不显示 Root Node

续上表

JTree 构造函数

JTree(Object[]value)

利用 Object Array 建立树, 不显示 Root Node

JTree(TreeModel newModel)

利用 TreeModel 建立树

JTree(TreeNode root)

利用 TreeNode 建立树

JTree(TreeNode root, boolean asksAllowsChildren)

利用 TreeNode 建立树, 并决定是否允许子节点的存在

JTree(Vector value)

利用 Vector 建立树, 不显示 Root Node

我们首先来看一下系统默认的树是什么样的:

范例 InitialTree.java (文件位于随书光盘目录 exam:ch10\InitialTree.java)

```

1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4
5  public class InitialTree
6  {
7      public InitialTree()
8      {
9          JFrame f = new JFrame("TreeDemo");
10         Container contentPane = f.getContentPane();
11
12         JTree tree = new JTree();
13         JScrollPane scrollPane = new JScrollPane();
14         scrollPane.setViewportView(tree);
15
16         contentPane.add(scrollPane);
17         f.pack();
18         f.setVisible(true);
19
20         f.addWindowListener(new WindowAdapter() {
21             public void windowClosing(WindowEvent e) {
22                 System.exit(0);
23             }
24         });
25
26     }
27
28     public static void main(String args[]) {
29
30         new InitialTree();
31     }
32 }

```

说明:

- (1) 这个程序相当简单, 我们的目的只是让读者看一下 Java 默认的树到底是什么样, 因此在程序第 12 行中, 我们建立一个空的 JTree 对象, 并将此 JTree 放入 JScrollPane 中, 使得当 JTree 的 size 大于窗口空间时, 可以利用滚动条来滚动面板。

程序运行结果如图 10-2 所示。



图 10-2

10-2 以 Hashtable 构造 JTree

上面这个例子对我们并没有实质的帮助, 因为各个节点的数据均是 Java 的默认值, 而非我们自己设置的。因此我们需利用其他 JTree 的构造函数来输入我们想要的节点数据。下面的范例我们以 Hashtable 当作 JTree 的数据输入:

范例 TreeDemo1.java (文件位于随书光盘目录 exam\ch10\TreeDemo1.java)

```

1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4  import java.util.*;
5
6  public class TreeDemo1
7  {
8      public TreeDemo1()
9      {
10         JFrame f = new JFrame("TreeDemo");
11         Container contentPane = f.getContentPane();
12
13         String[] s1 = {"公司文件", "个人信件", "私人文件"};
14         String[] s2 = {"本机磁盘 (C:)", "本机磁盘 (D:)", "本机磁盘 (E:)"};
15         String[] s3 = {"奇摩站", "职棒消息", "网络书店"};
16         Hashtable hashtable1 = new Hashtable();
17         Hashtable hashtable2 = new Hashtable();
18         hashtable1.put("我的公文包", s1);

```

```

19      hashtable1.put("我的电脑",s2);
20      hashtable1.put("收藏夹",hashtable2);
21      hashtable2.put("网站列表",s3);
22      JTree tree = new JTree(hashtable1);
23      JScrollPane scrollPane = new JScrollPane();
24      scrollPane.setViewportView(tree);
25
26      contentPane.add(scrollPane);
27      f.pack();
28      f.setVisible(true);
29
30      f.addWindowListener(new WindowAdapter() {
31          public void windowClosing(WindowEvent e) {
32              System.exit(0);
33          }
34      });
35
36  }
37
38  public static void main(String args[]) {
39
40      new TreeDemo1();
41  }
42  }

```

⊕ 说明:

- (1) Hashtable 是相当常用的数据结构,它是利用 key-value pair 的方式来存储数据的,也就是说当您要在 Hashtable 中找寻数据时,必须先提供 key 值,Hashtable 会根据 key 值找到所属的 value 值。而 Hashtable 是利用 put() 方法将 key-value 放入 Hashtable 中。程序第 18 行中,“我的公文包”就是一个 key 值,而 s1 String Array 就是“我的公文包”的 value 值。
- (2) 程序第 18~21 行表示此 JTree 的 Root Node 包含 3 个文件夹,分别是“我的公文包”、“我的电脑”与“收藏夹”,而“收藏夹”文件夹中又包含“网站列表”子文件夹。
- (3) 利用 JTree(Hashtable value)构造函数所建立的树是看不见根节点的。

⊕ 程序运行结果如图 10-3 所示。

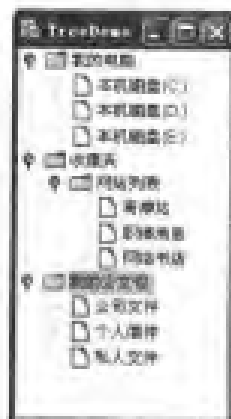


图 10-3

读者可利用类似的方式以 Vector 或 Object Array 来构造 JTree, 不过利用这两种构造函数所构造出来的 JTree 就不会像用 Hashtable 般的简单与美观, 我们将此部分留做习题。

10-3 以 TreeNode 构造 JTree

JTree 上的每一个节点就代表一个 TreeNode 对象, TreeNode 本身是一个 Interface, 里面定义了 7 种有关节点的方法, 例如判断是否为树叶节点 (isLeaf())、有几个子节点 (getChildCount())、父节点是谁 (getParent()) 等等, 这些方法的定义您可以在 javax.swing.tree 的 package 中找到, 读者可自行查阅 Java API 文件。在实际的应用上, 一般我们不会直接实作此界面, 而是采用 Java 所提供的 DefaultMutableTreeNode 类, 此类是实作 MutableTreeNode 界面而来, 并提供了其他许多实用的方法。MutableTreeNode 本身也是一个 Interface, 且继承了 TreeNode 界面, 此类主要是定义一些对节点的处理方式, 例如新增节点 (insert())、删除节点 (remove())、设置节点 (setUserObject()) 等。整个关系如图 10-4 所示。

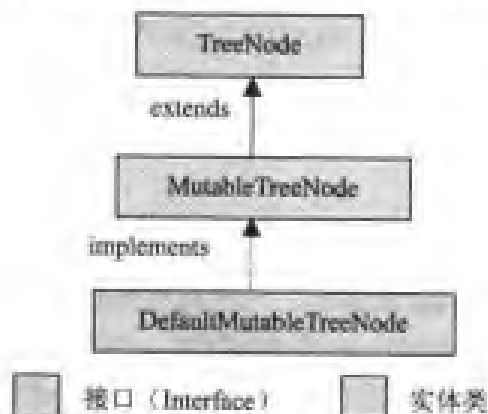


图 10-4

接下来我们来看如何利用 DefaultMutableTreeNode 来建立 JTree, 我们先来看 DefaultMutableTreeNode 的构造函数, 如表 10-2 所示。

表 10-2

DefaultMutableTreeNode 的构造函数	
DefaultMutableTreeNode()	建立空的 DefaultMutableTreeNode 对象
DefaultMutableTreeNode(Object userObject)	建立 DefaultMutableTreeNode 对象, 节点为 userObject 对象
DefaultMutableTreeNode(Object userObject, Boolean allowsChildren)	建立 DefaultMutableTreeNode 对象, 节点为 userObject 对象并决定此节点是否允许具有子节点

以下为利用 DefaultMutableTreeNode 建立 JTree 的范例。

范例 TreeDemo2.java (文件位于随书光盘目录 exam\ch10\TreeDemo2.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;

```

http://www.it-ebooks.info

```

4    import javax.swing.tree.*;
5
6    public class TreeDemo2
7    {
8        public TreeDemo2()
9        {
10            JFrame f = new JFrame("TreeDemo");
11            Container contentPane = f.getContentPane();
12
13            DefaultMutableTreeNode root =
14            new DefaultMutableTreeNode("资源管理器");
15            DefaultMutableTreeNode node1 =
16            new DefaultMutableTreeNode("我的公文包");
17            DefaultMutableTreeNode node2 =
18            new DefaultMutableTreeNode("我的电脑");
19            DefaultMutableTreeNode node3 =
20            new DefaultMutableTreeNode("收藏夹");
21            DefaultMutableTreeNode node4 =
22            new DefaultMutableTreeNode("Readme");
23            root.add(node1);
24            root.add(node2);
25            root.add(node3);
26            root.add(node4);
27
28            DefaultMutableTreeNode leafnode =
29            new DefaultMutableTreeNode("公司文件");
30            node1.add(leafnode);
31            leafnode = new DefaultMutableTreeNode("个人信件");
32            node1.add(leafnode);
33            leafnode = new DefaultMutableTreeNode("私人文件");
34            node1.add(leafnode);
35
36            leafnode = new DefaultMutableTreeNode("本机磁盘(C:)");
37            node2.add(leafnode);
38            leafnode = new DefaultMutableTreeNode("本机磁盘(D:)");
39            node2.add(leafnode);
40            leafnode = new DefaultMutableTreeNode("本机磁盘(E:)");
41            node2.add(leafnode);
42
43            DefaultMutableTreeNode node31 =
44            new DefaultMutableTreeNode("网站列表");
45            node3.add(node31);
46
47            leafnode = new DefaultMutableTreeNode("奇摩站");
48            node31.add(leafnode);
49            leafnode = new DefaultMutableTreeNode("职棒消息");
50            node31.add(leafnode);
51            leafnode = new DefaultMutableTreeNode("网络书店");
52            node31.add(leafnode);
53
54            JTree tree = new JTree(root);
55            JScrollPane scrollPane = new JScrollPane();
56            scrollPane.setViewportView(tree);
57
58            contentPane.add(scrollPane);

```

```

59         f.pack();
60         f.setVisible(true);
61
62         f.addWindowListener(new WindowAdapter() {
63             public void windowClosing(WindowEvent e) {
64                 System.exit(0);
65             }
66         });
67
68     }
69
70     public static void main(String args[]) {
71
72         new TreeDemo2();
73     }
74 }

```

⊕ 说明:

- (1) DefaultMutableTreeNode 为 javax.swing.tree package 中的一个类，因此在程序第 4 行中我们要将此 package 实作进来。
- (2) 利用 DefaultMutableTreeNode 的方法就是一层一层的包，例如在本例中节点“资源管理器”为根节点，它的子节点有 4 个，分别是“我的公文包”、“我的电脑”、“收藏夹”与“Readme”节点。其中“Readme”节点没有包含任何的子节点，因此它就是一个树叶节点。

⊕ 程序运行结果如图 10-5 所示。

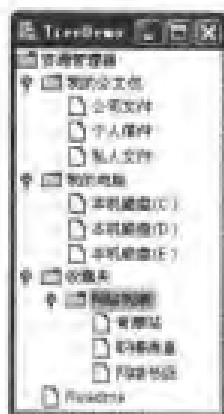


图 10-5

10-4 以 TreeModel 构造 JTree

除了以节点的观念(TreeNode)建立树之外，您可以以用 data model 的模式建立树。树的 data model 称为 TreeModel，用此模式的好处是可以触发树的相关事件，来处理树可能产生的一些变动。TreeModel 是一个 Interface，里面定义了 8 种方法：如果您是一个喜欢自己动手做的人，或是您想显示的数据格式很复杂，您可以考虑直接实作 TreeModel 接口中所定义的方法。

法来构造出 JTree。TreeModel 接口的方法如表 10-3 所示。

表 10-3

TreeModel 方法	
void	addTreeModelListener(TreeModelListener l) 增加一个 TreeModelListener 来监控 TreeModelEvent 事件
Object	getChild(Object parent, int index) 返回子节点
int	getChildCount(Object parent) 返回子节点数量
int	getIndexOfChild(Object parent, Object child) 返回子节点的索引值
Object	getRoot() 返回根节点
boolean	isLeaf(Object node) 判断是否为树叶节点
void	removeTreeModelListener(TreeModelListener l) 删除 TreeModelListener
void	valueForPathChanged(TreePath path, Object newValue) 当用户改变 Tree 上的值时如何应对

您可以实作出这 8 种方法，然后构造出自己想要的 JTree，不过在大部分的情况下我们通常不会这样做，毕竟要实作出这 8 种方法不是件轻松的事，而且 Java 本身也提供了一个默认模式类，叫做 DefaultTreeModel，这个类已经实现了 TreeModel 接口，也另外提供了许多实用的方法。利用这个默认模式，我们便能很方便的构造出 JTree 来了。表 10-4 为 DefaultTreeModel 的构造函数与范例。

表 10-4

DefaultTreeModel 的构造函数	
DefaultTreeModel(TreeNode root)	建立 DefaultTreeModel 对象，并定出根节点
DefaultTreeModel(TreeNode root, Boolean asksAllowsChildren)	建立具有根节点的 DefaultTreeModel 对象，并决定此节点是否允许具有子节点

范例 TreeDemo3.java (文件位于随书光盘目录 exam\ch10\TreeDemo3.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import javax.swing.tree.*;
5
6  public class TreeDemo3
7  {
8      public TreeDemo3()
9      {
10         JFrame f = new JFrame("TreeDemo");
11         Container contentPane = f.getContentPane();
12

```

```
13     DefaultMutableTreeNode root = new
14     DefaultMutableTreeNode("资源管理器");
15     DefaultMutableTreeNode node1 = new
16     DefaultMutableTreeNode("我的公文包");
17     DefaultMutableTreeNode node2 = new
18     DefaultMutableTreeNode("我的电脑");
19     DefaultMutableTreeNode node3 = new
20     DefaultMutableTreeNode("收藏夹");
21     DefaultMutableTreeNode node4 = new
22     DefaultMutableTreeNode("Readme");
23
24     DefaultTreeModel treeModel = new DefaultTreeModel(root);
25     treeModel.insertNodeInto(node1, root, root.getChildCount());
26     treeModel.insertNodeInto(node2, root, root.getChildCount());
27     treeModel.insertNodeInto(node3, root, root.getChildCount());
28     treeModel.insertNodeInto(node4, root, root.getChildCount());
29
30     DefaultMutableTreeNode leafnode = new
31     DefaultMutableTreeNode("公司文件");
32     treeModel.insertNodeInto(leafnode, node1, node1.getChildCount());
33     leafnode = new DefaultMutableTreeNode("个人信件");
34     treeModel.insertNodeInto(leafnode, node1, node1.getChildCount());
35     leafnode = new DefaultMutableTreeNode("私人文件");
36     treeModel.insertNodeInto(leafnode, node1, node1.getChildCount());
37
38     leafnode = new DefaultMutableTreeNode("本机磁盘(C:)");
39     treeModel.insertNodeInto(leafnode, node2, node2.getChildCount());
40     leafnode = new DefaultMutableTreeNode("本机磁盘(D:)");
41     treeModel.insertNodeInto(leafnode, node2, node2.getChildCount());
42     leafnode = new DefaultMutableTreeNode("本机磁盘(E:)");
43     treeModel.insertNodeInto(leafnode, node2, node2.getChildCount());
44
45     DefaultMutableTreeNode node31 = new
46     DefaultMutableTreeNode("网站列表");
47     treeModel.insertNodeInto(node31, node3, node3.getChildCount());
48     leafnode = new DefaultMutableTreeNode("奇摩站");
49     treeModel.insertNodeInto(leafnode, node3, node3.getChildCount());
50     leafnode = new DefaultMutableTreeNode("职棒消息");
51     treeModel.insertNodeInto(leafnode, node3, node3.getChildCount());
52     leafnode = new DefaultMutableTreeNode("网络书店");
53     treeModel.insertNodeInto(leafnode, node3, node3.getChildCount());
54
55     JTree tree = new JTree(treeModel);
56     JScrollPane scrollPane = new JScrollPane();
57     scrollPane.setViewportView(tree);
58
59     contentPane.add(scrollPane);
60     f.pack();
61     f.setVisible(true);
62
63     f.addWindowListener(new WindowAdapter() {
64         public void windowClosing(WindowEvent e) {
65             System.exit(0);
66         }
67     });
```

```

68
69     }
70
71     public static void main(String args[]) {
72
73         new TreeDemo3();
74     }
75 }

```

说明:

- (1) 程序第 24 行, 以根节点建立一个 DefaultTreeModel 对象。
 - (2) 程序第 25 行, 利用 DefaultTreeModel 类所提供的 insertNodeInto() 方法加入节点到父节点中。
 - (3) 程序第 25 行, 利用 DefaultMutableTreeNode 类所提供的 getChildCount() 方法取得目前子节点的数量。
 - (4) 程序第 55 行, 以 TreeModel 建立 JTree。
- 程序运行结果与上例相同。

10-5 改变 JTree 的外观

您可以使用 JComponent 所提供的 putClientProperty(Object key, Object value) 方法来设置 Java 默认的 JTree 外观。设置方式共有 3 种:

1. tree.putClientProperty("JTree.lineStyle", "None"): Java 默认值
2. tree.putClientProperty("JTree.lineStyle", "Horizontal"): 使 JTree 的文件夹间具有水平分隔线。
3. tree.putClientProperty("JTree.lineStyle", "Angled"): 使 JTree 具有类似 Windows 文件管理器的直角连接线。

若您想要设置此功能, 您可以在上例中的第 55~56 行间加入此行, 如:

```

JTree tree = new JTree(treeModel);
tree.putClientProperty("JTree.lineStyle", "Horizontal");
JScrollPane scrollPane = new JScrollPane();

```

则运行结果如图 10-6 所示。

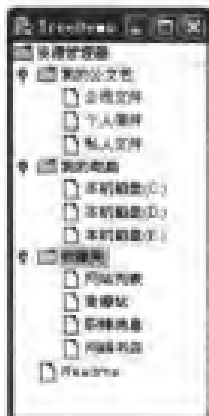


图 10-6

若改成:

```
JTree tree = new JTree(treeModel);
tree.putClientProperty("JTree.lineStyle", "Angled");
JScrollPane scrollPane = new JScrollPane();
```

⊕ 则运行结果如图 10-7 所示。

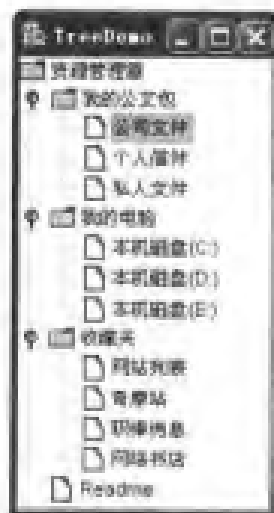


图 10-7

如果您已经习惯了 Windows 或 Unix 平台的操作系统,您可能会对 Java 所默认的 JTree 图案感到陌生,这时候您就可以使用 Java 所提供的 Look and Feel 功能,将 JTree 改成您习惯的平台图案。例如只要您在 TreeDemo3.java 中加入下列这几行,您就可以看到 Windows 平台的树状图标:

```
try {
    UIManager.setLookAndFeel(
        "com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
} catch (Exception e1) {
    System.exit(0);
}
```

⊕ 运行结果如下所图 10-8 所示。

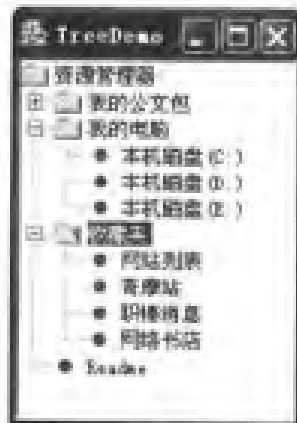


图 10-8

如果想改成 Unix 环境的图标, 您只要加入下面这几行程序代码:

```
try {
    UIManager.setLookAndFeel(
        "com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
} catch (Exception e1) {
    System.exit(0);
}
```

运行结果将如图 10-9 所示。



图 10-9

Java 的 Look and Feel 功能我们将在第 15 章中详细介绍。

10-6 更换 JTree 节点图案

JTree 利用 `TreeCellRenderer` 接口来运行绘制节点的工作, 同样的, 您不需要直接去实现这个接口所定义的方法, 因为 Java 本身提供一个已经实作好的类来给我们使用, 这个类就是 `DefaultTreeCellRenderer`, 您可以在 `javax.swing.tree` package 中找到此类所提供的方法。下面为使用 `DefaultTreeCellRenderer` 更改节点图案的一个例子:

范例 TreeDemo4.java (文件位于随书光盘目录 exam\ch10\TreeDemo4.java)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import javax.swing.tree.*;
5
6  public class TreeDemo4
7  {
8      public TreeDemo4()
9      {
10         JFrame f = new JFrame("TreeDemo");
11         Container contentPane = f.getContentPane();
12
13         DefaultMutableTreeNode root = new
14         DefaultMutableTreeNode("资源管理器");
```



```
15     DefaultMutableTreeNode node1 = new
16     DefaultMutableTreeNode("我的公文包");
17     DefaultMutableTreeNode node2 = new
18     DefaultMutableTreeNode("我的电脑");
19     DefaultMutableTreeNode node3 = new
20     DefaultMutableTreeNode("收藏夹");
21     DefaultMutableTreeNode node4 = new
22     DefaultMutableTreeNode("Readme");
23
24     DefaultTreeModel treeModel = new DefaultTreeModel(root);
25     treeModel.insertNodeInto(node1, root, root.getChildCount());
26     treeModel.insertNodeInto(node2, root, root.getChildCount());
27     treeModel.insertNodeInto(node3, root, root.getChildCount());
28     treeModel.insertNodeInto(node4, root, root.getChildCount());
29
30     DefaultMutableTreeNode leafnode = new
31     DefaultMutableTreeNode("公司文件");
32     treeModel.insertNodeInto(leafnode, node1, node1.getChildCount());
33     leafnode = new DefaultMutableTreeNode("个人信件");
34     treeModel.insertNodeInto(leafnode, node1, node1.getChildCount());
35     leafnode = new DefaultMutableTreeNode("私人文件");
36     treeModel.insertNodeInto(leafnode, node1, node1.getChildCount());
37
38     leafnode = new DefaultMutableTreeNode("本机磁盘(C:)");
39     treeModel.insertNodeInto(leafnode, node2, node2.getChildCount());
40     leafnode = new DefaultMutableTreeNode("本机磁盘(D:)");
41     treeModel.insertNodeInto(leafnode, node2, node2.getChildCount());
42     leafnode = new DefaultMutableTreeNode("本机磁盘(E:)");
43     treeModel.insertNodeInto(leafnode, node2, node2.getChildCount());
44
45     DefaultMutableTreeNode node31 = new
46     DefaultMutableTreeNode("网站列表");
47     treeModel.insertNodeInto(node31, node3, node3.getChildCount());
48     leafnode = new DefaultMutableTreeNode("奇摩站");
49     treeModel.insertNodeInto(leafnode, node3, node3.getChildCount());
50     leafnode = new DefaultMutableTreeNode("职棒消息");
51     treeModel.insertNodeInto(leafnode, node3, node3.getChildCount());
52     leafnode = new DefaultMutableTreeNode("网络书店");
53     treeModel.insertNodeInto(leafnode, node3, node3.getChildCount());
54
55     JTree tree = new JTree(treeModel);
56     tree.setRowHeight(20);
57
58     DefaultTreeCellRenderer cellRenderer =
59     (DefaultTreeCellRenderer)tree.getCellRenderer();
60
61     cellRenderer.setLeafIcon(new ImageIcon(".\\icons\\leaf.gif"));
62     cellRenderer.setOpenIcon(new ImageIcon(".\\icons\\open.gif"));
63     cellRenderer.setClosedIcon(new ImageIcon(".\\icons\\close.gif"));
64
65     cellRenderer.setFont(new Font("宋体", Font.PLAIN, 12));
66     cellRenderer.setBackgroundNonSelectionColor(Color.white);
67     cellRenderer.setBackgroundSelectionColor(Color.yellow);
68     cellRenderer.setBorderSelectionColor(Color.red);
69     cellRenderer.setTextNonSelectionColor(Color.black);
```

```

70         cellRenderer.setTextSelectionColor(Color.blue);
71
72         JScrollPane scrollPane = new JScrollPane();
73         scrollPane.setViewportView(tree);
74
75         contentPane.add(scrollPane);
76         f.pack();
77         f.setVisible(true);
78
79         f.addWindowListener(new WindowAdapter() {
80             public void windowClosing(WindowEvent e) {
81                 System.exit(0);
82             }
83         });
84
85     }
86
87     public static void main(String args[]) {
88
89         new TreeDemo4();
90     }
91 }

```

说明：

- (1) 程序第 56 行，设置每一行的高度。
- (2) 程序第 58~59 行，利用 JTree 的 getCellRenderer()方法取得 DefaultTreeCellRenderer 对象。
- (3) 程序第 61~63 行，分别设置目录打开、关闭与树叶节点的图案。
- (4) 程序第 65 行，设置字体。
- (5) 程序第 66~68 行，设置选中或未选中时，节点的底色颜色。
- (6) 程序第 69~70 行，设置选中或未选中时，文字的颜色。

运行结果如图 10-10 所示。

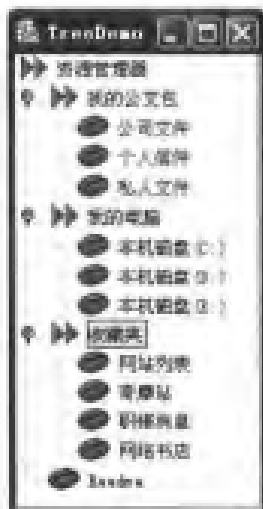


图 10-10

10-7 JTree 的事件处理模式

在此节中，我们将详细介绍 JTree 两个常用的事件处理，分别是 `TreeModeEvent` 与 `TreeSelectionEvent`。

10-7-1 处理 `TreeModeEvent` 事件

当树的结构有任何改变时，例如节点值改变了、新增了节点、删除了节点等，都会产生 `TreeModelEvent` 事件，要处理这样的事件必须实现 `TreeModelListener` 接口，此界面定义了 4 种方法，如表 10-5 所示。

表 10-5

TreeModelListener 的方法	
Void	<code>treeNodesChanged(TreeModelEvent e)</code> 当节点时改变时系统就会去调用这个方法
Void	<code>treeNodesInserted(TreeModelEvent e)</code> 新增节点时系统就会去调用这个方法
Void	<code>treeNodesRemoved(TreeModelEvent e)</code> 删除节点时系统就会去调用这个方法
Void	<code>treeStructureChanged(TreeModelEvent e)</code> 当树的结构改变时系统就会去调用这个方法

`TreeModelEvent` 类本身提供了 5 种方法，帮助我们取得事件的信息，如表 10-6 所示。

表 10-6

TreeModelEvent 的方法	
<code>int[]</code>	<code>getChildIndices()</code> 返回子节点群的索引值
<code>Object[]</code>	<code>getChildren()</code> 返回子节点群
<code>Object[]</code>	<code>getPath()</code> 返回 Tree 中一条 path 上（从 Root Node 到 Leaf Node）的节点
<code>TreePath</code>	<code>getTreePath()</code> 取得目前位置的 Tree Path
<code>String</code>	<code>toString()</code> 取得对象的字符串表示法

由 `TreeModelEvent` 的 `getTreePath()` 方法就可以得到 `TreePath` 对象，此对象能够让我们知道用户目前正点选哪一个节点，`TreePath` 类最常用的方法为：

`public Object getLastPathComponent()`：取得最深（内）层的节点。

`public int getPathCount()`：取得此 Path 上共有几个节点。

我们来看下面这个例子，用户可以在 Tree 上编辑节点，按下【Enter】键后就可以改变原有的值，并将改变的值显示在下面的 JLabel 中：

范例 TreeDemo5.java (文件位于随书光盘目录 exam\ch10\TreeDemo5.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import javax.swing.event.*;
5  import javax.swing.tree.*;
6
7  public class TreeDemo5 implements TreeModelListener
8  {
9      JLabel label = null;
10     String nodeName = null; //原有节点名称
11
12     public TreeDemo5()
13     {
14         JFrame f = new JFrame("TreeDemo");
15         Container contentPane = f.getContentPane();
16
17         DefaultMutableTreeNode root = new
18         DefaultMutableTreeNode("资源管理器");
19         DefaultMutableTreeNode node1 = new
20         DefaultMutableTreeNode("我的公文包");
21         DefaultMutableTreeNode node2 = new
22         DefaultMutableTreeNode("我的电脑");
23         DefaultMutableTreeNode node3 = new
24         DefaultMutableTreeNode("收藏夹");
25         DefaultMutableTreeNode node4 = new
26         DefaultMutableTreeNode("Readme");
27         root.add(node1);
28         root.add(node2);
29         root.add(node3);
30         root.add(node4);
31
32         DefaultMutableTreeNode leafnode = new
33         DefaultMutableTreeNode("公司文件");
34         node1.add(leafnode);
35         leafnode = new DefaultMutableTreeNode("个人信件");
36         node1.add(leafnode);
37         leafnode = new DefaultMutableTreeNode("私人文件");
38         node1.add(leafnode);
39
40         leafnode = new DefaultMutableTreeNode("本机磁盘(C:)");
41         node2.add(leafnode);
42         leafnode = new DefaultMutableTreeNode("本机磁盘(D:)");
43         node2.add(leafnode);
44         leafnode = new DefaultMutableTreeNode("本机磁盘(E:)");
45         node2.add(leafnode);
46
47         DefaultMutableTreeNode node31 = new
48         DefaultMutableTreeNode("网站列表");
49         node3.add(node31);
50
51         leafnode = new DefaultMutableTreeNode("奇摩站");
52         node31.add(leafnode);
53         leafnode = new DefaultMutableTreeNode("职棒消息");

```

```
54         node31.add(leafnode);
55         leafnode = new DefaultMutableTreeNode("网络书店");
56         node31.add(leafnode);
57
58         JTree tree = new JTree(root);
59         tree.setEditable(true);
60         tree.addMouseListener(new MouseHandle());
61         DefaultTreeModel treeModel = (DefaultTreeModel)tree.getModel();
62         treeModel.addTreeModelListener(this);
63
64         JScrollPane scrollPane = new JScrollPane();
65         scrollPane.setViewportView(tree);
66
67         label = new JLabel("更改数据为 : ");
68         contentPane.add(scrollPane, BorderLayout.CENTER);
69         contentPane.add(label, BorderLayout.SOUTH);
70         f.pack();
71         f.setVisible(true);
72
73         f.addWindowListener(new WindowAdapter() {
74             public void windowClosing(WindowEvent e) {
75                 System.exit(0);
76             }
77         });
78     }
79 }
80
81 public void treeNodesChanged(TreeModelEvent e) {
82
83     TreePath treePath = e.getTreePath();
84     DefaultMutableTreeNode node =
85         (DefaultMutableTreeNode)treePath.getLastPathComponent();
86     try {
87         int[] index = e.getChildIndices();
88         node = (DefaultMutableTreeNode)node.getChildAt(index[0]);
89     } catch (NullPointerException exc) {}
90     label.setText(nodeName+" 更改数据为 :
91 "+(String)node.getUserObject());
92 }
93 public void treeNodesInserted(TreeModelEvent e) {
94 }
95 public void treeNodesRemoved(TreeModelEvent e) {
96 }
97 public void treeStructureChanged(TreeModelEvent e) {
98 }
99
100 public static void main(String args[]) {
101
102     new TreeDemo5();
103 }
104
105 class MouseHandle extends MouseAdapter
106 {
107     public void mousePressed(MouseEvent e)
108     {
```

```

109         try{
110             JTree tree = (JTree)e.getSource();
111
112             int rowLocation = tree.getRowForLocation(e.getX(), e.getY());
113
114             TreePath treepath = tree.getPathForRow(rowLocation);
115             TreeNode treenode =
116             (TreeNode)treepath.getLastPathComponent();
117
118             nodeName = treenode.toString();
119             }catch(NullPointerException ne){}
120         }
121     }
122 }

```

⊕ 说明:

- (1) 程序第 59 行, 设置 JTree 为可编辑的。
- (2) 程序第 60 行, 使 Tree 加入检测 Mouse 事件, 以便取得节点名称。
- (3) 程序第 61~62 行, 取得 DefaultTreeModel, 并检测是否有 TreeModelEvent 事件。
- (4) 程序第 81~98 行, 实作 TreeModelListener 界面, 此界面共定义了四种方法, 分别是 treeNodesChanged()、treeNodesInserted()、treeNodesRemoved()、与 treeStructureChanged()。在此范例中我们只针对更改节点值的部分, 因此只实作 treeNodesChanged()方法。
- (5) 程序第 84~85 行, 由 TreeModelEvent 取得的 DefaultMutableTreeNode 为节点的父节点, 而不是用户点选的节点, 这点读者要特别注意。要取得真正的节点需再加写第 86~89 行。
- (6) 程序第 87 行, getChildIndices()方法会返回目前修改节点的索引值。由于我们只修改一个节点, 因此节点索引值就放在 index[0]的位置。若点选的节点为 Root Node, 则 getChildIndices()的返回值为 null, 程序第 89 行就是处理点选 Root Node 产生的 NullPointerException 问题。
- (7) 程序第 88 行, 由 DefaultMutableTreeNode 类的 getChildAt()方法取得修改的节点对象。
- (8) 程序第 91 行, 由 DefaultMutableTreeNode 类的 getUserObject()方法取得节点内容, 或是写成 node.toString()也相同。
- (9) 程序第 105~121 行, 处理 Mouse 点选事件。
- (10) 程序第 112 行, JTree 的 getRowForLocation()方法会返回节点的列索引值。例如下图中, “本机磁盘(D:)”的列索引值为 4, 此索引值会随着其他文件夹的打开或收起而变动, 但“资源管理器”的列索引值永为 0。
- (11) 程序第 114 行, JTree 的 getPathForRow()方法会取得从 Root Node 到点选节点的一条 Path, 此 Path 为一条直线, 如下图若您点选“本机磁盘(E)”, 则 Tree Path 为“资源管理器”-->“网上邻居”-->“本机磁盘(E)”。因此利用 TreePath 的 getLastPathComponent()方法就可以取得所点选的节点。

(12) 您也可以直接将 112~114 行写成 `TreePath treepath = tree.getSelectionPath()`, 意义相同。

◆ 程序运行结果如图 10-11 所示。



图 10-11

我们将“我的电脑”改成“网上邻居”。

我们再来看一个 `TreeModelEvent` 的例子, 下面这个例子我们可以让用户自行增加、删除与修改节点:

范例 TreeDemo6.java (文件位于随书光盘目录 exam1ch10\TreeDemo6.java)

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import javax.swing.event.*;
5  import javax.swing.tree.*;
6
7  public class TreeDemo6 implements ActionListener, TreeModelListener
8  {
9      JLabel label = null;
10     JTree tree = null;
11     DefaultTreeModel treeModel = null;
12     String nodeName = null; //原有节点名称
13
14     public TreeDemo6()
15     {
16         JFrame f = new JFrame("TreeDemo");
17         Container contentPane = f.getContentPane();
18
19         DefaultMutableTreeNode root = new
20         DefaultMutableTreeNode("资源管理器");
21
22         tree = new JTree(root);
23         tree.setEditable(true);
```

```

24         tree.addMouseListener(new MouseHandle());
25         treeModel = (DefaultTreeModel)tree.getModel();
26         treeModel.addTreeModelListener(this);
27
28         JScrollPane scrollPane = new JScrollPane();
29         scrollPane.setViewPortView(tree);
30
31         JPanel panel = new JPanel();
32         JButton b = new JButton("新增节点");
33         b.addActionListener(this);
34         panel.add(b);
35         b = new JButton("删除节点");
36         b.addActionListener(this);
37         panel.add(b);
38         b = new JButton("清除所有节点");
39         b.addActionListener(this);
40         panel.add(b);
41
42         label = new JLabel("Action");
43         contentPane.add(panel, BorderLayout.NORTH);
44         contentPane.add(scrollPane, BorderLayout.CENTER);
45         contentPane.add(label, BorderLayout.SOUTH);
46         f.pack();
47         f.setVisible(true);
48
49         f.addWindowListener(new WindowAdapter() {
50             public void windowClosing(WindowEvent e) {
51                 System.exit(0);
52             }
53         });
54
55     }
56
57     public void actionPerformed(ActionEvent ae)
58     {
59         if (ae.getActionCommand().equals("新增节点"))
60         {
61             DefaultMutableTreeNode parentNode = null;
62             DefaultMutableTreeNode newNode = new
63             DefaultMutableTreeNode("新节点");
64             newNode.setAllowsChildren(true);
65             TreePath parentPath = tree.getSelectionPath();
66
67             parentNode = (DefaultMutableTreeNode)
68                 (parentPath.getLastPathComponent());
69
70             treeModel.insertNodeInto(
71                 newNode, parentNode, parentNode.getChildCount());
72             tree.scrollPathToVisible(new TreePath(newNode.getPath()));
73
74             label.setText("新增节点成功");

```



```

75         }
76
77         if (ae.getActionCommand().equals("删除节点"))
78         {
79             TreePath treepath = tree.getSelectionPath();
80             if (treepath != null)
81             {
82                 DefaultMutableTreeNode selectionNode =
83                     (DefaultMutableTreeNode) treepath.getLastPathComponent();
84
85                 TreeNode parent = (TreeNode) selectionNode.getParent();
86                 if (parent != null)
87                 {
88                     treeModel.removeNodeFromParent(selectionNode);
89                     label.setText("删除节点成功");
90                 }
91             }
92         }
93
94         if (ae.getActionCommand().equals("清除所有节点"))
95         {
96             DefaultMutableTreeNode rootNode =
97                 (DefaultMutableTreeNode) treeModel.getRoot();
98             rootNode.removeAllChildren();
99             treeModel.reload();
100            label.setText("清除所有节点成功");
101        }
102    }
103
104    public void treeNodesChanged(TreeModelEvent e) {
105
106        TreePath treePath = e.getTreePath();
107        DefaultMutableTreeNode node =
108            (DefaultMutableTreeNode) treePath.getLastPathComponent();
109        try {
110            int[] index = e.getChildIndices();
111            node = (DefaultMutableTreeNode) node.getChildAt(index[0]);
112        } catch (NullPointerException exc) {}
113        label.setText(nodeName+" 更改数据为 :
114        "+(String) node.getUserObject());
115    }
116    public void treeNodesInserted(TreeModelEvent e) {
117        System.out.println("new node inserted");
118    }
119    public void treeNodesRemoved(TreeModelEvent e) {
120        System.out.println("node deleted");
121    }
122    public void treeStructureChanged(TreeModelEvent e) {
123        System.out.println("structrue changed");
124    }
125

```

```

126     public static void main(String args[]) {
127
128         new TreeDemo6();
129     }
130
131     class MouseHandle extends MouseAdapter
132     {
133         public void mousePressed(MouseEvent e)
134         {
135             try{
136                 JTree tree = (JTree)e.getSource();
137
138                 int rowLocation = tree.getRowForLocation(
139 e.getX(), e.getY());
140                 TreePath treepath = tree.getPathForRow(rowLocation);
141                 TreeNode treenode = (TreeNode)
142 treepath.getLastPathComponent();
143
144                 nodeName = treenode.toString();
145             }catch(NullPointerException ne){}
146         }
147     }
148 }
149

```

⊕ 说明:

- (1) 程序第 57~102 行, 运行新增、删除、清除所有节点的程序代码。
- (2) 程序第 59~75 行, 运行新增节点程序代码。
- (3) 程序第 64 行, `DefaultMutableTreeNode` 的 `setAllowsChildren()` 方法使产生的新节点可以包含子节点。
- (4) 程序第 65~68 行, 目的在取得新节点的父节点。
- (5) 程序第 70~71 行, 由 `DefaultTreeModel` 的 `insertNodeInto()` 方法增加新节点。
- (6) 程序第 72 行, `JTree` 的 `scrollPathToVisible()` 方法使 `Tree` 会自动展开文件夹以便显示所加入的新节点。若没加这行则加入的新节点会被包在文件夹中, 您必须自行展开文件夹才看得到。
- (7) 程序第 77~92 行, 运行删除节点程序代码。
- (8) 程序第 82~85 行, 取得所选取节点的父节点。
- (9) 程序第 88 行, 由 `DefaultTreeModel` 的 `removeNodeFromParent()` 方法删除节点, 包括它的子节点。
- (10) 程序第 94~101 行, 运行清除所有节点的程序代码。
- (11) 程序第 96~97 行, 由 `DefaultTreeModel` 的 `getRoot()` 方法取得根节点。
- (12) 程序第 98 行, 删除所有子节点。
- (13) 程序第 99 行, 删除完后务必运行 `DefaultTreeModel` 的 `reload()` 操作, 整个 `Tree` 的节点才会真正被删除。

◆ 程序运行结果如下：

程序初始状态，如图 10-12 所示。



图 10-12

新增节点，如图 10-13 所示。



图 10-13

当您新增删除节点，或是清除所有节点时，您可以在 DOS 命令行中看到这些信息，如图 10-14 所示。

```
new node inserted  
new node inserted  
new node inserted  
new node inserted  
new node inserted  
node deleted  
structure changed
```

图 10-14

您会发现，当您新增节点后，系统会自动调用 `treeNodesInserted()` 方法；当您删除节点后，系统会自动调用 `treeNodesRemoved()` 方法；当您清除所有节点后，因为运行了 `DefaultTreeModel` 的 `reload()` 操作，因此系统会自动调用 `treeStructureChanged()` 方法。

10-7-2 处理 TreeSelectionEvent 事件

当我们在 JTree 上點選任何一个节点时，都会触发 TreeSelectionEvent 事件，如果我们要处理这样的事件，就必须实现 TreeSelectionListener 接口，此界面只定义了一种方法，那就是 valueChanged() 方法。

TreeSelectionEvent 常用于处理显示节点的内容，例如您在文件图标中点两下就可以看到文件的内容。在 JTree 中选择节点的方式共有 3 种，这 3 种情况跟选择 JList 上的项目是一模一样的，分别是：

DISCONTIGUOUS_TREE_SELECTION：可作单一选择，连续节点选择（按住【Shift】键），不连续选择多个节点（按住【Ctrl】键），这是 Java 的默认值。

CONTINUOUS_TREE_SELECTION：按住【Shift】键，可对某一连续的节点区间作选取。

SINGLE_TREE_SELECTION：一次只能选一个节点。

您可以自行实作 TreeSelectionModel 制作出更复杂的选择方式，但通常是没必要的，因为 Java 提供了默认的选择模式类供我们使用，那就是 DefaultTreeSelectionModel，利用这个类我们可以很方便的设置上面 3 种选择模式。

下面这个范例，当用户點選了一个文件名称时，就会将文件的内容显示出来：

范例 TreeDemo7.java (文件位于随书光盘目录 exam\ch10\TreeDemo7.java)

```

1  import javax.swing.*;
2  import javax.swing.tree.*;
3  import javax.swing.event.*;
4  import java.awt.*;
5  import java.awt.event.*;
6  import java.io.*;
7
8  public class TreeDemo7 implements TreeSelectionListener
9  {
10     JEditorPane editorPane;
11
12     public TreeDemo7()
13     {
14         JFrame f = new JFrame("TreeDemo");
15         Container contentPane = f.getContentPane();
16         DefaultMutableTreeNode root = new
17         DefaultMutableTreeNode("资源管理器");
18         DefaultMutableTreeNode node = new
19         DefaultMutableTreeNode("TreeDemo1.java");
20         root.add(node);
21         node = new DefaultMutableTreeNode("TreeDemo2.java");
22         root.add(node);
23         node = new DefaultMutableTreeNode("TreeDemo3.java");
24         root.add(node);
25         node = new DefaultMutableTreeNode("TreeDemo4.java");
26         root.add(node);
27
28         JTree tree = new JTree(root);
29         tree.getSelectionModel().setSelectionMode(

```

```

30         TreeSelectionMode.SINGLE_TREE_SELECTION);
31     tree.addTreeSelectionListener(this);
32
33     JScrollPane scrollPanel = new JScrollPane(tree);
34     editorPane = new JEditorPane();
35     JScrollPane scrollPane2 = new JScrollPane(editorPane);
36     JSplitPane splitPane = new JSplitPane(
37         JSplitPane.HORIZONTAL_SPLIT,true, scrollPanel, scrollPane2);
38
39     contentPane.add(splitPane);
40     f.pack();
41     f.setVisible(true);
42
43     f.addWindowListener(new WindowAdapter() {
44         public void windowClosing(WindowEvent e) {
45             System.exit(0);
46         }
47     });
48 }
49
50 public void valueChanged(TreeSelectionEvent e)
51 {
52     JTree tree = (JTree) e.getSource();
53     DefaultMutableTreeNode selectionNode =
54         (DefaultMutableTreeNode)tree.getLastSelectedPathComponent();
55
56     String nodeName = selectionNode.toString();
57
58     if (selectionNode.isLeaf())
59     {
60         String filepath = "file:"+System.getProperty("user.dir") +
61             System.getProperty("file.separator") +
62             nodeName;
63
64         try {
65             editorPane.setPage(filepath);
66         } catch (IOException ex) {
67             System.out.println("找不到此文件");
68         }
69     }
70 }
71
72 public static void main(String[] args) {
73     new TreeDemo7();
74 }
75 }

```

说明:

- (1) 程序第 29~30 行, 设置 Tree 的选择模式为一次只能选择一个节点。
- (2) 程序第 31 行, 检测是否有 TreeSelectionEvent 事件。
- (3) 程序第 33~37 行, 在 JSplitPane 中, 左边是放含有 JTree 的 JScrollPane, 右边是放 JEditorPane。

- (4) 程序第 50~70 行, 实作 `TreeSelectionListener` 的 `valueChanged()` 方法。
- (5) 程序第 53~54 行, 利用 `JTree` 的 `getLastSelectedPathComponent()` 方法取得目前选取的节点。
- (6) 程序第 58~69 行, 判断是否为树叶节点, 若是则显示文件内容, 若不是则不做任何操作。
- (7) 程序第 60~62 行, 取得文件的位置路径。`System.getProperty("user.dir")` 可以取得目前工作的路径, `System.getProperty("file.separator")` 是取得文件分隔符, 例如在 Windows 环境的文件分隔符是 “\”, 而 Unix 环境的文件分隔符刚好相反, 是 “/”。利用 `System.getProperty()` 方法您可以取得下列的信息:

<code>java.version</code>	显示 Java 版本
<code>java.vendor</code>	显示 Java 制造商
<code>java.vendor.url</code>	显示 Java 制造商 URL
<code>java.home</code>	显示 Java 的安装路径
<code>java.class.version</code>	显示 Java 类版本
<code>java.class.path</code>	显示 Java classpath
<code>os.name</code>	显示操作系统名称
<code>os.arch</code>	显示操作系统结构, 如 x86
<code>os.version</code>	显示操作系统版本
<code>file.separator</code>	取得文件分隔符
<code>path.separator</code>	取得路径分隔符, 如 Unix 是以 “:” 表示
<code>line.separator</code>	取得换行符号, 如 Unix 是以 “\n” 表示
<code>user.name</code>	取得用户名称
<code>user.home</code>	取得用户家目录 (home directory), 如在 Windows 中 Administrator 的家目录为 C:\Documents and Settings\Administrator
<code>user.dir</code>	取得用户当前的工作目录

- (8) 程序第 65 行, 利用 `JEditorPane` 的 `setPage()` 方法将文件内容显示在 `editorPane` 中。
- 若文件路径错误, 则会产生 `IOException`。
- 程序运行结果如图 10-15 所示。

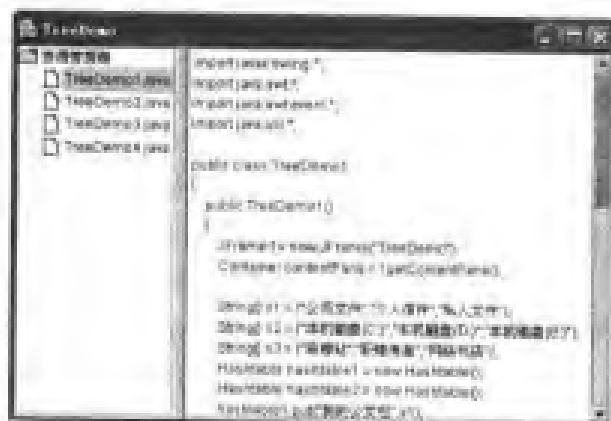


图 10-15

10-8 JTree 的其他操作

在上面各小节中,我们已经介绍完 JTree 大部分的功能,这些也是 JTree 最常用到的操作方式。不过还有一些小功能是我们没讲到的,这些功能虽然不常用到,但我们还是将它提出来,让有兴趣的读者自行参考相关信息。

我们在之前的小节中曾说到 Tree 中的每一个节点都是一个 `TreeNode`,并可利用 JTree 的 `setEditable()`方法设置节点是否可编辑。若要在 Tree 中寻找节点的父节点或子节点,或判断是否为树叶节点,皆可由实作 `TreeNode` 界面做到,但要编辑节点呢? Java 将编辑节点的任务交给 `TreeCellEditor`,`TreeCellEditor` 本身是一个界面,里面只定义了 `getTreeCellEditorComponent()`方法,您可以实作此方法使节点具有编辑的效果。不过您不用这么辛苦去实作这个方法,Java 本身提供了 `DefaultTreeCellEditor` 类来实作此方法,亦提供了其他许多方法,例如取得节点内容(`getCellEditorValue()`)、设置节点字体(`setFont()`)、决定节点是否可编辑(`isCellEditable()`)等等。除非您觉得 `DefaultTreeCellEditor` 所提供的功能还不够,您才需要去实作 `TreeCellEditor` 界面。您可以利用 JTree 的 `getCellEditor()`方法取得 `DefaultTreeCellEditor` 对象。当我们编辑节点时会触发 `ChangeEvent` 事件,您可以实现 `CellEditorListener` 接口来处理此事件,`CellEditorListener`接口包括两种方法,分别是 `editingStopped(ChangeEvent e)`与 `editingCanceled(ChangeEvent e)`,若您没有实作 `TreeCellEditor` 接口,系统会以默认的 `DefaultTreeCellEditor` 类来处理这两种方法(您可以在 `DefaultTreeCellEditor` 中找到这两种方法),因此您无须再编写任何程序。

另外,JTree 还有一种事件处理模式,那就是 `TreeExpansionEvent` 事件。要处理这个事件您必须实现 `TreeExpansionListener` 界面,此接口定义了两种方法,分别是 `treeCollapsed(TreeExpansionEvent e)`与 `treeExpanded(TreeExpansionEvent e)`。当节点展开时系统就会自动调用 `treeExpanded()`方法,当节点合起来时,系统就会自动调用 `treeCollapsed()`方法。您可以在这两种方法中编写所要处理事件的程序代码。处理事件的过程我们已在前面例子中举过多次,相信读者应该不陌生才是,我们将此部分留做习题。

10-9 本章总结

Tree 是 Swing 才有的组件,在本章中我们一开始提到如何以最简单的方式来构造 JTree,后面提到以 `Hashtable` 来构造 JTree。介绍完比较简单的 JTree 构造方式后,我们接着介绍如何以 `TreeNode` 与 `TreeMode` 来构造 JTree,使用这两种方式来构造 JTree,我们就能做出比较复杂的操作,例如找到节点的父节点或子节点、设置节点图案、改变节点外观等。最后我们介绍了 JTree 的事件处理模式,例如可在改变节点值时做其他效果,或是选择节点时显示出节点的内容等,或是动态的新增或删除节点。在看完此章的同时,您应该会感觉到 Swing 组件几乎都有 `Model` 的存在,而且一个组件可能有好几种 `Model` 来决定组件的显示效果,并有 Java 提供的默认 `Model` 方便我们使用,这说明了 Swing 本身是包含在 MVC (`Model View Controller`) 的模式中。

10-10

本章习题

1. 改写 TreeDemo1.java 程序, 利用 Vector 来构造 JTree, 使之外观如图 10-16 所示。

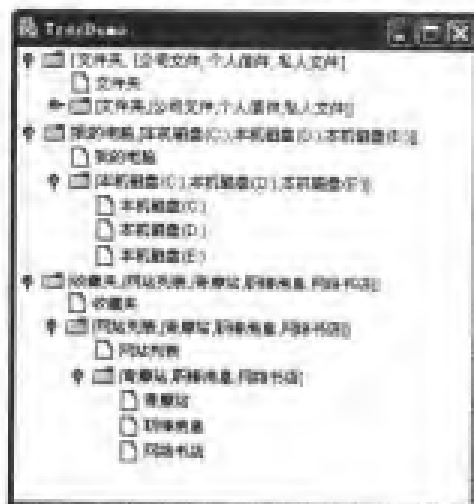


图 10-16

2. 改写 TreeDemo7.java 程序, 使之能输出 HTML 格式的文件内容。(Hint: 使用 setPage(URL page))。
3. 改写 TreeDemo7.java 程序, 利用处理 MouseEvent 事件, 使得当鼠标在节点上双击时才会显示文件内容。
4. 实作 TreeExpansionListener 界面, 使得当节点展开或收起时会有“哔”的一声音效。
5. 试述 JTree 各种 Model 的功能与使用方法。

11

对话框 (Option Pane 与 Dialog) 的使用与介绍

如果您要为公司设计一套“物品工具借用系统”，在这个系统中，借用物品的员工必须详填员工编号、借用器具、借用日期、预计归还日期时间、借用原因等等，若没有详细填写这些数据，就无法取得准许借用物品的证明文件。因此在设计这套系统的过程中，您必须察看用户是否已经填妥相关数据，若员工忘了填写某些重要字段，系统应该给予警示，提醒用户哪些字段必须填写。善用对话框可以增进系统与用户之间的沟通，并在必要时提供给用户相关信息，帮助用户了解系统状况。在本章中，我们将详细地介绍如何使用 JDialog 与 JOptionPane，并说明它们之间的差异与使用时机。

11-1 使用 JDialog 组件

JDialog 的类层次结构图:

```
java.lang.Object
--java.awt.Component
  --java.awt.Container
    --java.awt.Window
      --java.awt.Dialog
        -- javax.swing.JDialog
```

如果您要为公司设计一套“物品工具借用系统”，在这个系统中，借用物品的员工必须详填员工编号、借用器具、借用日期、预计归还日期时间、借用原因等等，若没有详细填写这些数据，就无法取得准许借用物品的证明文件。因此在设计这套系统的过程中，您必须察看用户是否已经填妥相关数据，若员工忘了填写某些重要字段，系统应该给予警示，提醒用户哪些字段必须填写。这个情况也常发生在网络问卷或网络会员注册系统上，用户必须填写相关数据，例如用户若没有填写 E-mail 地址，则系统会提示你应当填写邮件地址，否则系统将不处理用户填写的信息。

为应对这种情况，Java 提供了 JDialog 与 JOptionPane 供我们使用。事实上，JOptionPane 是阳春版的 JDialog，当您在使用 JOptionPane 时，系统会自动产生 JDialog 组件，并将 JOptionPane 的内容放入 JDialog 的 ContentPane 中，而这些均由系统在后台自动运行，并不需要我们介入。使用 JOptionPane 的好处是此组件已经默认了许多交互方式，您只用设置想要的显示模式，JOptionPane 就能轻易的显示出来，可以说是相当方便，若这些模式还是无法满足您的需求，您就可以使用 JDialog 来自行设计您的对话框。

我们先来看如何构造 JDialog，JOptionPane 将在后半段介绍。表 11-1 是 JDialog 的构造函数。

表 11-1

JDialog 的构造函数

JDialog()	建立一个 non-modal 的对话框，没有 title 也不属于任何事窗组件
JDialog(Dialog owner)	建立一个属于 Dialog 组件的对话框，为 non-modal 形式，也没有 title
JDialog(Dialog owner, Boolean modal)	建立一个属于 Dialog 组件的对话框，可决定 modal 形式，但没有 title
JDialog(Dialog owner, String title)	建立一个属于 Dialog 组件的对话框，为 non-modal 形式，对话框上有 title
JDialog(Dialog owner, String title, Boolean modal)	建立一个属于 Dialog 组件的对话框，可决定 modal 形式，且对话框上有 title
JDialog(Frame owner)	建立一个属于 Frame 组件的对话框，为 non-modal 形式，也没有 title
JDialog(Frame owner, Boolean modal)	建立一个属于 Frame 组件的对话框，可决定 modal 形式，但没有 title

续上表

JDialog 的构造函数

JDialog(Frame owner, String title)

建立一个属于 Frame 组件的对话框, 为 non-modal 形式, 对话框上有 title

JDialog(Frame owner, String title, Boolean modal)

建立一个属于 Frame 组件的对话框, 可决定 modal 形式, 且对话框上有 title

上面所说的 modal 是一种对话框操作模式, 当 modal 为 true 时, 代表用户必须结束对话框才能回到原来所属的窗口。当 modal 为 false 时, 代表对话框与所属窗口可以互相切换, 彼此之间在操作上没有顺序性。

一般而言对话框都会依附在某个窗口上, 例如 JFrame 或 JDialog, 原因在于对话框通常是一个程序在运行的过程中与用户互动的中间过程。在使用 JDialog 上跟使用 JFrame 非常类似, 由上面的 JDialog 层次结构图您可以发现, JDialog 是继承 AWT 的 Dialog 类而来, 因此 JDialog 为一个 Heavyweight 组件。要加入组件到 JDialog 上与 JFrame 是一样的, 您必须先取得 JDialog 的 ContentPane, 然后再把组件加到此 ContentPane 中, JDialog 默认的版面管理器是 BorderLayout。

11-1-1 在 JFrame 上建立 JDialog

我们来看一个 JDialog 的例子。在这个例子中, 用户在主窗口按下“借用物品”按钮时, 会跳出一个让用户填写相关信息的 JDialog 窗口, 用户必须结束此 JDialog 窗口后才能返回主窗口。

范例 DialogDemo.java (文件位于随书光盘目录 exam\ch11\DialogDemo.java)

```

1  import javax.swing.*;
2  import javax.swing.border.*;
3  import java.awt.*;
4  import java.awt.event.*;
5
6  public class DialogDemo implements ActionListener
7  {
8      JFrame f = null;
9
10     public void actionPerformed(ActionEvent e) {
11         String cmd = e.getActionCommand();
12         if (cmd.equals("借用物品")) {
13             new LendingSystem(f);
14         } else if (cmd.equals("离开系统")) {
15             System.exit(0);
16         }
17     }
18     public DialogDemo()
19     {
20         f = new JFrame("JDialog Example");
21         Container contentPane = f.getContentPane();
22         JPanel buttonPanel = new JPanel();
23         JButton b = new JButton("借用物品");

```

```

24         b.addActionListener(this);
25         buttonPanel.add(b);
26         b = new JButton("离开系统");
27         b.addActionListener(this);
28         buttonPanel.add(b);
29
30         buttonPanel.setBorder(BorderFactory.createTitledBorder(
31             BorderFactory.createLineBorder(Color.blue,2),
32             "借用物品系统",TitledBorder.CENTER,TitledBorder.TOP));
33
34         contentPane.add(buttonPanel, BorderLayout.CENTER);
35         f.pack();
36         f.setVisible(true);
37
38         f.addWindowListener(new WindowAdapter() {
39             public void windowClosing(WindowEvent e) {
40                 System.exit(0);
41             }
42         });
43     }
44
45     public static void main(String[] args)
46     {
47         new DialogDemo();
48     }
49 }
50
51 class LendingSystem implements ActionListener
52 {
53     JTextField staffField,objectField,
54     borrowDateField,returnDateField,reasonField;
55     JDialog dialog;
56
57     public void actionPerformed(ActionEvent e) {
58         String cmd = e.getActionCommand();
59         if (cmd.equals("确定")) {
60             }
61         else if(cmd.equals("取消")){
62             dialog.dispose();
63         }
64     }
65
66     LendingSystem(JFrame f){
67         dialog = new JDialog(f,"借用物品",true);
68         GridBagConstraints c;
69         int gridx,gridy,gridwidth,gridheight,anchor,fill,ipadx,ipady;
70         double weightx,weighty;
71         Insets inset;
72
73         GridBagLayout gridbag = new GridBagLayout();
74         Container dialogPane = dialog.getContentPane();
75         dialogPane.setLayout(gridbag);
76
77         JLabel label = new JLabel("员工编号 : ");
78         gridx=0;           //第 0 列

```

```

79         gridy=0;                //第0行
80         gridwidth = 1;           //占一单位宽度
81         gridheight = 1;          //占一单位高度
82         weightx = 0;             //窗口增大时组件宽度增大比率 0
83         weighty = 0;             //窗口增大时组件高度增大比率 0
84         anchor = GridBagConstraints.CENTER; //容器大于组件 size 时
85 //将组件置于容器中央
86         fill = GridBagConstraints.BOTH; //窗口拉大时会
87 //填满水平与垂直空间
88         inset = new Insets(0,0,0,0); //组件间间距
89         ipadx = 0;                //组件内水平宽度
90         ipady = 0;                //组件内垂直高度
91         c = new GridBagConstraints(gridx,gridy,gridwidth,gridheight,
92 weightx,weighty,anchor,fill,inset,ipadx,ipady);
93         gridbag.setConstraints(label,c);
94         dialogPane.add(label);
95
96         label = new JLabel("借用器具 : ");
97         gridx=3;
98         gridy=0;
99         c = new GridBagConstraints(gridx,gridy,gridwidth,gridheight,
100 weightx,weighty,anchor,fill,inset,ipadx,ipady);
101         gridbag.setConstraints(label,c);
102         dialogPane.add(label);
103
104         label = new JLabel("借用日期 : ");
105         gridx=0;
106         gridy=1;
107         c = new GridBagConstraints(gridx,gridy,gridwidth,gridheight,
108 weightx,weighty,anchor,fill,inset,ipadx,ipady);
109         gridbag.setConstraints(label,c);
110         dialogPane.add(label);
111
112         label = new JLabel("归还日期 : ");
113         gridx=3;
114         gridy=1;
115         c = new GridBagConstraints(gridx,gridy,gridwidth,gridheight,
116 weightx,weighty,anchor,fill,inset,ipadx,ipady);
117         gridbag.setConstraints(label,c);
118         dialogPane.add(label);
119
120         label = new JLabel("借用原因 : ");
121         gridx=0;
122         gridy=2;
123         c = new GridBagConstraints(gridx,gridy,gridwidth,gridheight,
124 weightx,weighty,anchor,fill,inset,ipadx,ipady);
125         gridbag.setConstraints(label,c);
126         dialogPane.add(label);
127
128         staffField = new JTextField();
129         gridx=1;
130         gridy=0;
131         gridwidth = 2;
132         gridheight = 1;
133         weightx = 1;

```

```
134         weighty = 0;
135         c = new GridBagConstraints(gridx, gridy, gridwidth, gridheight,
136             weightx, weighty, anchor, fill, inset, ipadx, ipady);
137         gridbag.setConstraints(staffField, c);
138         dialogPane.add(staffField);
139
140         objectField = new JTextField();
141         gridx=4;
142         gridy=0;
143         c = new GridBagConstraints(gridx, gridy, gridwidth, gridheight,
144             weightx, weighty, anchor, fill, inset, ipadx, ipady);
145         gridbag.setConstraints(objectField, c);
146         dialogPane.add(objectField);
147
148         borrowDateField = new JTextField();
149         gridx=1;
150         gridy=1;
151         c = new GridBagConstraints(gridx, gridy, gridwidth, gridheight,
152             weightx, weighty, anchor, fill, inset, ipadx, ipady);
153         gridbag.setConstraints(borrowDateField, c);
154         dialogPane.add(borrowDateField);
155
156         returnDateField = new JTextField();
157         gridx=4;
158         gridy=1;
159         c = new GridBagConstraints(gridx, gridy, gridwidth, gridheight,
160             weightx, weighty, anchor, fill, inset, ipadx, ipady);
161         gridbag.setConstraints(returnDateField, c);
162         dialogPane.add(returnDateField);
163
164         reasonField = new JTextField();
165         gridx=1;
166         gridy=2;
167         gridwidth = 5;
168         c = new GridBagConstraints(gridx, gridy, gridwidth, gridheight,
169             weightx, weighty, anchor, fill, inset, ipadx, ipady);
170         gridbag.setConstraints(reasonField, c);
171         dialogPane.add(reasonField);
172
173         JPanel panel = new JPanel();
174         panel.setLayout(new GridLayout(1, 2));
175         JButton b = new JButton("确定");
176         panel.add(b);
177         b = new JButton("取消");
178         b.addActionListener(this);
179         panel.add(b);
180
181         gridx=0;
182         gridy=3;
183         gridwidth = 6;
184         weightx = 1;
185         weighty = 1;
186         c = new GridBagConstraints(gridx, gridy, gridwidth, gridheight,
187             weightx, weighty, anchor, fill, inset, ipadx, ipady);
188         gridbag.setConstraints(panel, c);
```

```

189         dialogPane.add(panel);
190
191         dialog.setBounds(200,150,400,130);
192         dialog.show();
193     }
194 }

```

◆ 说明:

- (1) 程序第 10~17 行, 处理用户在主窗口中按下按钮的事件。当用户按下“借用物品”按钮时, 会产生一个 LendingSystem 对象, 此对象会 show 出一个 JDialog 窗口。
 - (2) 程序第 57~64 行, 处理用户在 JDialog 中按下按钮的事件。在此我们不处理用户按下“确定”按钮的事件。当用户按下“取消”按钮时, 就会运行 dispose() 方法, 此方法可让 JDialog 窗口关闭, 并释放系统资源。
 - (3) 程序第 67 行, 我们产生一个依附在 JFrame f 上的 JDialog 窗口, 并设置 modal 为 true, 代表我们需先关闭 JDialog 窗口才能回到 JFrame f 上。
- 程序运行结果如图 11-1 所示。



图 11-1

当按下“借用物品”按钮时, 如图 11-2 所示。



图 11-2

而此时的主窗口会呈现 Inactive 状态, 您必须关闭 JDialog 窗口, 才能再点选主窗口。

11-1-2 在 JApplet 上建立 JDialog

看完上个例子, 您也许会有疑问, 在 JDialog 的构造函数中, 只能依附在 Dialog 或 Frame 中, 那是不是 JDialog 不能用在 Applet 上呢? 答案是否定的, 您可以利用 Component 类所提供的 getParent() 方法, 找到 JApplet 所属的 Frame 容器, 这样就能在 JApplet 上显示出 JDialog 了。如下所示:

范例 DialogAppletDemo.java (文件位于随书光盘目录 exam\ch11\DialogAppletDemo.java)

```

1  /*
2   * <Applet code=DialogAppletDemo.class width=300 height=100>
3   * </Applet>
4   */
5
6  import javax.swing.*;
7  import javax.swing.border.*;
8  import java.awt.*;
9  import java.awt.event.*;
10
11 public class DialogAppletDemo extends JApplet implements ActionListener
12 {
13     Frame f = null;
14     public void actionPerformed(ActionEvent e) {
15         String cmd = e.getActionCommand();
16         if (cmd.equals("借用物品")) {
17             new LendingSystem(f);
18         } else if (cmd.equals("离开系统")) {
19             System.exit(0);
20         }
21     }
22     public void init()
23     {
24         Container contentPane = getContentPane();
25         for(Container c = this; c != null; c = c.getParent())
26         {
27             if (c instanceof Frame)
28                 f = (Frame)c;
29         }
30         JPanel buttonPanel = new JPanel();
31         JButton b = new JButton("借用物品");
32         b.addActionListener(this);
33         buttonPanel.add(b);
34         b = new JButton("离开系统");
35         b.addActionListener(this);
36         buttonPanel.add(b);
37
38         buttonPanel.setBorder(BorderFactory.createTitledBorder(
39             BorderFactory.createLineBorder(Color.blue, 2),
40             "借用物品系统", TitledBorder.CENTER, TitledBorder.TOP));
41
42         contentPane.add(buttonPanel, BorderLayout.CENTER);
43     }
44 }
45
46 class LendingSystem implements ActionListener
47 {
48     JTextField staffField, objectField,
49     borrowDateField, returnDateField, reasonField;
50     JDialog dialog;
51
52     public void actionPerformed(ActionEvent e) {
53         String cmd = e.getActionCommand();

```

```

54         if (cmd.equals("确定")) {
55             }
56         else if(cmd.equals("取消")){
57             dialog.dispose();
58         }
59     }
60
61     LendingSystem(Frame f){
62         dialog = new JDialog(f,"借用物品",true);
63         GridBagConstraints c;
64         int gridx,gridy,gridwidth,
65             gridheight,anchor,fill,ipadx,ipady;
66         double weightx,weighty;
67         Insets inset;
68         .....
69         .....
70         以下程序省略，与上面范例相同！
71     }

```

说明：

- (1) 这个例子是我们改写成的 Applet 形式，因此不再有 main 这个程序区块，取而代之的是 init 这个程序区块。
- (2) 利用 getParent()取得的 JApplet 父容器是 Frame 形式，而不是 JFrame 形式。因此在程序第 13 与 61 行，必须改成 Frame 的数据类型。
- (3) 程序第 27 行，instanceof 操作数的作用在于判断两边的数据类型是否相同，若相同则返回 true。

程序运行结果如图 11-3 所示。



图 11-3

当按下“借用物品”按钮时，如图 11-4 所示。



图 11-4

若您在程序第 26~27 行间加入 `System.out.println(c.getName());` 此行, 则在程序运行时, 您会在 DOS 下看到下面的信息, 如图 11-5 所示。

```
D:\nczhi\11>appletviewer DialogAppletDemo.java
名称: pane10
名称: pane11
名称: frame0
```

图 11-5

显示出 JApplet 上面的父类容器中, 有一个 Frame 的容器。

11-2 使用 JOptionPane 类的静态方法

JOptionPane 的类层次结构图:

```
java.lang.Object
--java.awt.Component
--java.awt.Container
--javax.swing.JComponent
--javax.swing.JOptionPane
```

以 JDialog 来制作对话框, 您必须实作对话框中的每一个组件, 但有时候我们的对话框只是要显示一段文字, 或是一些简单的选择 (是或否), 这时候可以利用 JOptionPane 类, 它可以让您很简单的做出这样的效果, 不仅大大地减少了程序代码的编写, 也让整个程序看起来简捷许多。

表 11-2 是 JOptionPane 的构造函数:

表 11-2

JOptionPane 的构造函数	
JOptionPane()	建立一个显示测试信息的 JOptionPane 组件
JOptionPane(Object message)	建立一个显示特定信息的 JOptionPane 组件
JOptionPane(Object message, int messageType)	建立一个显示特定信息的 JOptionPane 组件, 并设置信息类型
JOptionPane(Object message, int messageType, int optionType)	建立一个显示特定信息的 JOptionPane 组件, 并设置信息与选项类型
JOptionPane(Object message, int messageType, int optionType, Icon icon)	建立一个显示特定信息的 JOptionPane 组件, 并设置信息与选项类型, 且可以显示出图案
JOptionPane(Object message, int messageType, int optionType, Icon icon, Object[] options)	建立一个显示特定信息的 JOptionPane 组件, 并设置信息与选项类型, 且可以显示出图案。选项值是一个 Object Array, 可用作更改按钮上的文字
JOptionPane(Object message, int messageType, int optionType, Icon icon, Object[] options, Object initialValue)	建立一个显示特定信息的 JOptionPane 组件, 并设置信息与选项类型, 且可以显示出图案。选项值是一个 Object Array, 可用作更改按钮上的文字, 并设置默认按钮

对话框(Option Pane 与 Dialog)的使用与介绍

1 1

使用 JOptionPane 对象所得到的对话框都是 modal 为 true 的形式，也就是说我们必须先关闭对话框才能回到产生对话框的母窗口上。

要利用 JOptionPane 类来输出对话框，通常我们不会新建一个 JOptionPane 对象出来，而是使用 JOptionPane 所提供的一些静态方法 (Static Method)，不用产生 JOptionPane 对象就可以直接使用，这些方法都是以 showXxxxxDialog 的形式出现，若您的对话框是出现在 InternalFrame 上，您可以用 showInternalXxxxxDialog 的各种方法产生对话框。以下我们整理出 JOptionPane 提供输出对话框的所有静态方法，如表 11-3 至表 11-6 所示。

表 11-3

Message Dialog	
方法	void showMessageDialog(Component parentComponent, Object message) void showMessageDialog (Component parentComponent, Object message, String title, int messageType) void showMessageDialog(Component parentComponent, Object message, String title, int messageType, Icon icon) void showInternalMessageDialog(Component parentComponent, Object message) void showInternalMessageDialog(Component parentComponent, Object message, String title, int messageType) void showInternalMessageDialog(Component parentComponent, Object message, String title, int messageType, Icon icon)
说明	<p>显示信息对话框，对话框中只含有一个按钮，通常是“确定”按钮，例如安装完某个软件时通常会跳出一个对话框告知您安装已经成功。这类的方法有 5 种参数：</p> <p>parentComponent: 是指产生对话框的组件为何，通常是指 Frame 或 Dialog 组件</p> <p>message: 是指要显示的组件，通常是 String 或 Label 类型</p> <p>title: 对话框标题栏上显示的文字</p> <p>messageType: 指定信息类型，共有 5 种类型，分别是 ERROR_MESSAGE, INFORMATION_MESSAGE, WARNING_MESSAGE, QUESTION_MESSAGE, PLAIN_MESSAGE (不显示图标)。指定类型后对话框就会出现相对应的图标</p> <p>icon: 若您不喜欢 Java 给的图标，您可以自己自定图标</p>

表 11-4

Confirm Dialog	
方法	int showConfirmDialog(Component parentComponent, Object message) int showConfirmDialog(Component parentComponent, Object message, String title, int optionType) int showConfirmDialog(Component parentComponent, Object message, String title, int optionType, int messageType) int showConfirmDialog(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon) int showInternalConfirmDialog(Component parentComponent, Object message) int showInternalConfirmDialog(Component parentComponent, Object message, String title, int optionType) int showInternalConfirmDialog(Component parentComponent, Object message, String title, int optionType, int messageType) int showInternalConfirmDialog(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon)

续上表

Confirm Dialog	
说明	<p>显示确认对话框，这类对话框通常会问用户一个问题，然后用户回答是或不是。例如当我们修改了某个文件的内容却没有存盘就要离开时，系统大部分都会弹出确认对话框，询问我们是否要存储修改过的内容。确认对话框的方法有 6 种参数：</p> <p>parentComponent: 是指产生对话框的组件为何，通常是指 Frame 或 Dialog 组件</p> <p>message: 是指要显示的组件，通常是 String 或 Label 类型</p> <p>title: 对话框标题栏上显示的文字</p> <p>optionType: 确定按钮的类型，有 5 种类型，分别是 DEFAULT_OPTION、YES_NO_OPTION、YES_NO_CANCEL_OPTION、与 OK_CANCEL_OPTION</p> <p>messageType: 指定信息类型，共有 5 种类型，分别是 ERROR_MESSAGE、INFORMATION_MESSAGE、WARNING_MESSAGE、QUESTION_MESSAGE、PLAIN_MESSAGE。指定类型后对话框就会出现相对应的图标</p> <p>icon: 若您不喜欢 Java 给的图标，您可以自定义图标</p> <p>返回值为一整数值，依用户按下什么按钮而定，YES_OPTION = 0, NO_OPTION = 1, CANCEL_OPTION = 2, OK_OPTION = 0, CLOSED_OPTION = -1 (当用户都不选直接关掉对话框时)</p>

表 11-5

Input Dialog	
方法	<p>String showInputDialog(Object message)</p> <p>String showInputDialog(Component parentComponent, Object message)</p> <p>String showInputDialog(Component parentComponent, Object message, String title, int messageType)</p> <p>Object showInputDialog(Component parentComponent, Object message, String title, int messageType, Icon icon, Object[] selectionValues, Object initialSelectionValue)</p> <p>String showInternalInputDialog(Object message)</p> <p>String showInternalInputDialog(Component parentComponent, Object message)</p> <p>String showInternalInputDialog(Component parentComponent, Object message, String title, int messageType)</p> <p>Object showInternalInputDialog(Component parentComponent, Object message, String title, int messageType, Icon icon, Object[] selectionValues, Object initialSelectionValue)</p>
说明	<p>显示输入对话框，这类对话框可以让用户输入相关的信息，当用户按下确定按钮后，系统会得到用户所输入的信息。输入对话框不仅可以让用户自行输入文字，也可以提供 Combo Box 组件让用户选择相关信息，避免用户输入错误。输入对话框的方法有 6 种参数：</p> <p>parentComponent: 是指产生对话框的组件为何，通常是指 Frame 或 Dialog 组件</p> <p>message: 是指要显示的组件，通常是 String 或 Label 类型</p> <p>title: 对话框标题栏上显示的文字</p> <p>messageType: 指定信息类型，共有 5 种类型，分别是 ERROR_MESSAGE、INFORMATION_MESSAGE、WARNING_MESSAGE、QUESTION_MESSAGE、PLAIN_MESSAGE。指定类型后对话框就会出现相对应的图标</p> <p>icon: 若您不喜欢 Java 给的图标，您可以自定义图标</p> <p>selectionValue: 给用户选择的可能值。Object Array 中的数据会以 ComboBox 方式显示出</p> <p>initialSelectionValue: 对话框初始化时所显示的值</p> <p>当用户按下确定按钮时会返回用户输入的信息，若按下取消按钮则返回 null</p>

表 11-6

Option Dialog	
方法	<code>int showOptionDialog(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon, Object[] options, Object initialValue)</code> <code>int showInternalOptionDialog(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon, Object[] options, Object initialValue)</code>
说明	<p>显示选择对话框，这类对话框可以让用户自定义对话框类型，最大的好处是可以改变按钮上的文字。选择对话框的方法有 6 种参数：</p> <p>parentComponent: 是指产生对话框的组件为何，通常是指 Frame 或 Dialog 组件</p> <p>message: 是指要显示的组件，通常是 String 或 Label 类型</p> <p>title: 对话框标题栏上显示的文字</p> <p>optionType: 确定按钮的类型，有 5 种类型，分别是 DEFAULT_OPTION、YES_NO_OPTION、YES_NO_CANCEL_OPTION、与 OK_CANCEL_OPTION</p> <p>messageType: 指定信息类型，共有 5 种类型，分别是 ERROR_MESSAGE、INFORMATION_MESSAGE、WARNING_MESSAGE、QUESTION_MESSAGE、PLAIN_MESSAGE。指定类型后对话框就会出现相对应的图标</p> <p>icon: 若您不喜欢 Java 给的图标，您可以自定义图标</p> <p>options: 给用户选择的按钮显示文字</p> <p>initialValue: 对话框初始化时按钮的默认值</p> <p>返回值为一整数值，依用户按下什么按钮而定，YES_OPTION = 0, NO_OPTION = 1, CANCEL_OPTION = 2, OK_OPTION = 0, CLOSED_OPTION = -1 (当用户都不选直接关掉对话框时)</p>

上面的表格看起来好像很多方法似的，但实际上只区分成四大类，您只需要选择要用哪类对话框，再决定使用那类中的哪个方法即可。我们慢慢来为您介绍这四类对话框。

11-2-1 输出 Message Dialog

Message Dialog 是在对话框上显示出一段信息，目的在于告知用户一些相关信息，因此 Message Dialog 只会有一个确定按钮，让用户看完信息后就可以关闭这个对话框。下面这个例子中我们使用 Message 对话框，我们来看看不同的 MessageType 会有什么样的图案产生：

范例 MessageDialog.java (文件位于随书光盘目录 exam\ch11\MessageDialog.java)

```

1  import java.awt.*;
2  import javax.swing.*;
3  import java.awt.event.*;
4
5  public class MessageDialog implements ActionListener
6  {
7      JFrame f = null;
8
9      public MessageDialog()
10     {
11         f = new JFrame("OptionPane Demo");
12         Container contentPane = f.getContentPane();
13         contentPane.setLayout(new GridLayout(2,3));
14
15         JButton b = new JButton("Show Error Icon");

```

```
16         b.addActionListener(this);
17         contentPane.add(b);
18         b = new JButton("Show Information Icon");
19         b.addActionListener(this);
20         contentPane.add(b);
21         b = new JButton("Show Warning Icon");
22         b.addActionListener(this);
23         contentPane.add(b);
24         b = new JButton("Show Question Icon");
25         b.addActionListener(this);
26         contentPane.add(b);
27         b = new JButton("Show Plain Icon");
28         b.addActionListener(this);
29         contentPane.add(b);
30         b = new JButton("Show User Define Icon");
31         b.addActionListener(this);
32         contentPane.add(b);
33
34         f.pack();
35         f.setVisible(true);
36
37         f.addWindowListener(new WindowAdapter() {
38             public void windowClosing(WindowEvent e) {
39                 System.exit(0);
40             }
41         });
42     }
43
44     public static void main(String[] args)
45     {
46         new MessageDialog();
47     }
48
49     public void actionPerformed(ActionEvent e)
50     {
51         String cmd = e.getActionCommand();
52         String title = "Message Dialog";
53         String message = "";
54         int type = JOptionPane.PLAIN_MESSAGE;
55
56         if(cmd.equals("Show Error Icon")) {
57             type = JOptionPane.ERROR_MESSAGE;
58             message = " Error Message";
59         } else if(cmd.equals("Show Information Icon")) {
60             type = JOptionPane.INFORMATION_MESSAGE;
61             message = " Information Message";
62         } else if(cmd.equals("Show Warning Icon")) {
63             type = JOptionPane.WARNING_MESSAGE;
64             message = " Warning Message";
65         } else if(cmd.equals("Show Question Icon")) {
66             type = JOptionPane.QUESTION_MESSAGE;
67             message = " Question Message";
68         } else if(cmd.equals("Show Plain Icon")) {
69             type = JOptionPane.PLAIN_MESSAGE;
70             message = " Plain Message";
```

```

71         } else if(cmd.equals("Show User Define Icon")) {
72             type = JOptionPane.PLAIN_MESSAGE;
73             message = " User Define Message";
74             JOptionPane.showMessageDialog(f, message, title,
75                                         type, new ImageIcon("glass.jpg"));
76             return;
77         }
78
79         JOptionPane.showMessageDialog(f, message, title, type);
80     }
81 }

```

说明:

- (1) 程序第 49~80 行, 处理用户按钮事件, 若用户选择的是“Show User Define Icon”按钮, 则输出自定义 Icon 的信息对话框, 否则输出 Java 所提供的各种类型信息对话框。
- (2) 程序第 79 行, 利用 JOptionPane 的 showMessageDialog() 方法输出 Message Dialog。程序运行结果如图 11-6 所示。



图 11-6

按下“Show Error Icon”按钮时, 如图 11-7 所示。

按下“Show Information Icon”按钮时, 如图 11-8 所示。



图 11-7



图 11-8

按下“Show Warning Icon”按钮时, 如图 11-9 所示。

按下“Show Question Icon”按钮时, 如图 11-10 所示。



图 11-9



图 11-10

按下“Show Plain Icon”按钮时, 如图 11-11 所示。

按下“Show User Define Icon”按钮时, 如图 11-12 所示。



图 11-11



图 11-12

您会看出利用不同的信息类型，就可以显示出不同的信息图案。

11-2-2 输出 Confirm Dialog

看过了信息对话框后，接着我们来看确认对话框到底长怎样。Confirm Dialog 的目的在于让用户对某个问题选择“Yes”或“No”，可算是一种相当简单的是非选择对话框。下面这个范例中，用户可选取不同按钮类型的确认对话框，为方便说明，在此我们将 Message Type 都默认为 JOptionPane.INFORMATION_MESSAGE。

范例 ConfirmDialog.java (文件位于随书光盘目录 exam\ch11\ConfirmDialog.java)

```

1  import java.awt.*;
2  import javax.swing.*;
3  import java.awt.event.*;
4
5  public class ConfirmDialog implements ActionListener
6  {
7      JFrame f = null;
8      JLabel label = null;
9
10     public ConfirmDialog()
11     {
12         f = new JFrame("OptionPane Demo");
13         Container contentPane = f.getContentPane();
14
15         JPanel panel = new JPanel();
16         panel.setLayout(new GridLayout(2,2));
17
18         JButton b = new JButton("Show DEFAULT_OPTION");
19         b.addActionListener(this);
20         panel.add(b);
21         b = new JButton("Show YES_NO_OPTION");
22         b.addActionListener(this);
23         panel.add(b);
24         b = new JButton("Show YES_NO_CANCEL_OPTION");
25         b.addActionListener(this);
26         panel.add(b);
27         b = new JButton("Show OK_CANCEL_OPTION");
28         b.addActionListener(this);
29         panel.add(b);
30
31         label = new JLabel(" ", JLabel.CENTER);
32         contentPane.add(label, BorderLayout.NORTH);
33         contentPane.add(panel, BorderLayout.CENTER);
34         f.pack();

```

```

35         f.setVisible(true);
36
37         f.addWindowListener(new WindowAdapter() {
38             public void windowClosing(WindowEvent e) {
39                 System.exit(0);
40             }
41         });
42     }
43
44     public static void main(String[] args)
45     {
46         new ConfirmDialog();
47     }
48
49     public void actionPerformed(ActionEvent e)
50     {
51         String cmd = e.getActionCommand();
52         String title = "Confirm Dialog";
53         String message = "";
54         int messageType = JOptionPane.INFORMATION_MESSAGE;
55         int optionType = JOptionPane.YES_NO_OPTION;
56
57         if(cmd.equals("Show DEFAULT_OPTION")) {
58             optionType = JOptionPane.DEFAULT_OPTION;
59             message = "Show DEFAULT_OPTION Buttons";
60         } else if(cmd.equals("Show YES_NO_OPTION")) {
61             optionType = JOptionPane.YES_NO_OPTION;
62             message = "Show YES_NO_OPTION Buttons";
63         } else if(cmd.equals("Show YES_NO_CANCEL_OPTION")) {
64             optionType = JOptionPane.YES_NO_CANCEL_OPTION;
65             message = "Show YES_NO_CANCEL_OPTION Buttons";
66         } else if(cmd.equals("Show OK_CANCEL_OPTION")) {
67             optionType = JOptionPane.OK_CANCEL_OPTION;
68             message = "Show OK_CANCEL_OPTION Buttons";
69         }
70
71         int result = JOptionPane.showConfirmDialog(f, message,
72             title, optionType, messageType);
73
74         if (result == JOptionPane.YES_OPTION)
75             label.setText("您选择: Yes or OK");
76         if (result == JOptionPane.NO_OPTION)
77             label.setText("您选择: No");
78         if (result == JOptionPane.CANCEL_OPTION)
79             label.setText("您选择: Cancel");
80         if (result == JOptionPane.CLOSED_OPTION)
81             label.setText("您没做任何选择, 并关闭了对话框");
82     }
83 }

```

◆ 说明:

- (1) 程序第 49~82 行, 处理用户按钮事件, 默认的 messageType 是 JOptionPane.INFORMATION_MESSAGE。

Java Swing 程序设计

(2) 程序第 71~72 行, 利用 `JOptionPane` 的 `showConfirmDialog()` 方法输出 `Confirm Dialog`。当用户在确认对话框中按下按钮或关闭对话框时, 会返回一个整数值来判断用户到底做了什么操作。

✎ 程序运行结果如图 11-13 所示。



图 11-13

当按下 “Show DEFAULT_OPTION” 按钮时, 如图 11-14 所示。



图 11-14

当按下 “Show YES_NO_OPTION” 按钮时, 如图 11-15 所示。



图 11-15

当按下 “Show YES_NO_CANCEL_OPTION” 按钮时, 如图 11-16 所示。



图 11-16

当按下 “Show OK_CANCEL_OPTION” 按钮时, 如图 11-17 所示。

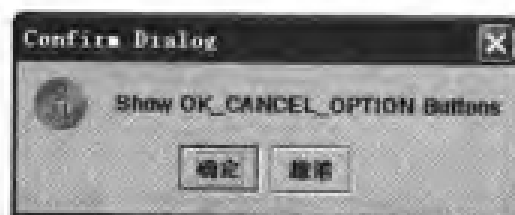


图 11-17

当用户点选了“确定”按钮时，会在主窗口的按钮上方出现“您选择：Yes or OK”的字样，如图 11-18 所示。



图 11-18

11-2-3 输出 Input Dialog

Input Dialog 可以让用户输入相关信息，当用户按下确定钮后，系统会得到用户所输入的信息。输入对话框不仅可以让用户自行输入文字，也可以显示出 ComboBox 组件让用户选择相关信息，避免用户输入错误。当用户输入完毕按下确定钮时会返回用户输入的信息，若按下取消钮则返回 null 值。下面为 InputDialog 的范例：

范例 InputDialog.java (文件位于随书光盘目录 exam\ch11\InputDialog.java)

```

1  import java.awt.*;
2  import javax.swing.*;
3  import java.awt.event.*;
4
5  public class InputDialog implements ActionListener
6  {
7      JFrame f = null;
8      JLabel label = null;
9
10     public InputDialog()
11     {
12         f = new JFrame("OptionPane Demo");
13         Container contentPane = f.getContentPane();
14
15         JPanel panel = new JPanel();
16         panel.setLayout(new GridLayout(1,2));
17
18         JButton b = new JButton("Show Text Input");
19         b.addActionListener(this);
20         panel.add(b);
21         b = new JButton("Show ComboBox Input");
22         b.addActionListener(this);
23         panel.add(b);
24
25         label = new JLabel(" ",JLabel.CENTER);
26         contentPane.add(label,BorderLayout.NORTH);
27         contentPane.add(panel,BorderLayout.CENTER);
28         f.pack();
29         f.setVisible(true);
30
31         f.addWindowListener(new WindowAdapter() {
32             public void windowClosing(WindowEvent e) {
33                 System.exit(0);
34             }
35         });
36     }
37 }

```

```

35         });
36     }
37
38     public static void main(String[] args)
39     {
40         new InputDialog();
41     }
42
43     public void actionPerformed(ActionEvent e)
44     {
45         String cmd = e.getActionCommand();
46         String title = "Input Dialog";
47         String message = "您最熟悉哪一种程序语言? ";
48         int messageType = JOptionPane.QUESTION_MESSAGE;
49         String[] values = {"JAVA", "PHP", "ASP", "C++", "VB"};
50         String result = "";
51
52         if(cmd.equals("Show Text Input")) {
53             result = JOptionPane.showInputDialog(f, message,
54                 title, messageType);
55
56         } else if(cmd.equals("Show ComboBox Input")) {
57             result = (String)JOptionPane.showInputDialog(f, message,
58                 title, messageType, null, values, values[0]);
59         }
60
61         if (result == null)
62             label.setText("您取消了对话框");
63         else{
64             label.setText("您输入: "+result);
65         }
66     }
67 }

```

◆ 说明:

- (1) 程序第 43~66 行, 处理用户按钮事件, 默认的 messageType 是 JOptionPane.QUESTION_MESSAGE。
- (2) 程序第 52~54 行, 当用户按下“Show Text Input”按钮时, 利用 JOptionPane 的 showConfirmDialog()方法输出一个含有 TextField 的 Input Dialog。当用户在此 Input Dialog 中按下“确定”按钮时, 会返回一个 String Object, 若按下“撤消”按钮则返回 null 值。
- (3) 程序第 56~59 行, 当用户按下“Show ComboBox Input”按钮时, 可利用 JOptionPane 的 showConfirmDialog()方法输出一个含有 ComboBox 的 Input Dialog, 并设“Java”(values[0])为 ComboBox 的系统默认值。

程序运行结果如图 11-19 所示。



图 11-19

当用户按下“Show Text Input”按钮时，如图 11-20 所示。



图 11-20

当用户按下“Show ComboBox Input”按钮时，如图 11-21 所示。



图 11-21

当用户选取完并按下“确定”按钮时，会在主窗口的按钮上方出现选取信息，如图 11-22 所示。



图 11-22

11-2-4 输出 Option Dialog

Option Dialog 可以让用户自定义对话框类型，比较有弹性，最大的好处是可以改变按钮上的文字。我们来看下面的例子：

范例 OptionDialog.java (文件位于随书光盘目录 exam\ch11\OptionDialog.java)

```
1  import java.awt.*;
2  import javax.swing.*;
3  import java.awt.event.*;
4
5  public class OptionDialog implements ActionListener
6  {
7      JFrame f = null;
8      JLabel label = null;
9
10     public OptionDialog()
11     {
12         f = new JFrame("OptionPane Demo");
13         Container contentPane = f.getContentPane();
14
15         JButton b = new JButton("Show Option Dialog");
```

```
16         b.addActionListener(this);
17
18         label = new JLabel(" ", JLabel.CENTER);
19         contentPane.add(label, BorderLayout.NORTH);
20         contentPane.add(b, BorderLayout.CENTER);
21         f.pack();
22         f.setVisible(true);
23
24         f.addWindowListener(new WindowAdapter() {
25             public void windowClosing(WindowEvent e) {
26                 System.exit(0);
27             }
28         });
29     }
30
31     public static void main(String[] args)
32     {
33         new OptionDialog();
34     }
35
36     public void actionPerformed(ActionEvent e)
37     {
38         String title = "Option Dialog";
39         String message = "您喜欢吃汉堡吗? ";
40         int messageType = JOptionPane.QUESTION_MESSAGE;
41         int optionType = JOptionPane.YES_NO_CANCEL_OPTION;
42         String[] options = {"喜欢", "不喜欢", "取消"};
43
44         int result = JOptionPane.showOptionDialog(f, message, title,
45             optionType, messageType, null, options, options[1]);
46
47         if (result == JOptionPane.YES_OPTION)
48             label.setText("您选择: 喜欢");
49         if (result == JOptionPane.NO_OPTION)
50             label.setText("您选择: 不喜欢");
51         if (result == JOptionPane.CANCEL_OPTION)
52             label.setText("您选择: 取消");
53         if (result == JOptionPane.CLOSED_OPTION)
54             label.setText("您没做任何选择, 并关闭了对话框");
55     }
56 }
```

⊕ 说明:

- (1) 程序第 36~55 行, 处理用户按钮事件, 默认的 messageType 是 JOptionPane.QUESTION_MESSAGE。
- (2) 程序第 44~45 行, 利用 JOptionPane 的 showOptionDialog() 方法输出 Option Dialog。当用户在确认对话框中按下按钮或关闭对话框时, 会返回一个整数值来判断用户到底做了什么操作。
- (3) 由于我们的 optionType 设置成 JOptionPane.YES_NO_CANCEL_OPTION, 因此对话框中会有三个按钮。我们在 options 的 String Array 中设置这三个按钮的名称, 并

以 options[1] 为按钮默认值。若将 options 参数设为 null，系统会用原来的按钮名称来显示。

程序运行结果如图 11-23 所示。



图 11-23

当按下“Show Option Dialog”按钮时，如图 11-24 所示。



图 11-24

当用户点选了“不喜欢”按钮时，会在主窗口的按钮上方出现“您选择：不喜欢”的字样，如图 11-25 所示。



图 11-25

11-2-5 输出 Internal Dialog

我们之前曾经说过，JOptionPane 也可以显示出 Internal Dialog 对话框，使用方法跟上面的范例一模一样，只是在方法名称上多了 Internal 这个字眼，例如 showInternalMessageDialog() 等等。我们来看下面的范例：

范例 InternalDialog.java (文件位于随书光盘目录 exam\ch11\InternalDialog.java)

```
1  import java.awt.*;
2  import javax.swing.*;
3  import java.awt.event.*;
4
5  public class InternalDialog implements ActionListener
6  {
7      JInternalFrame internalFrame = null;
8      JLabel label = null;
9
10     public InternalDialog()
```



```
11     {
12         JFrame f = new JFrame("OptionPane Demo");
13         Container contentPane = f.getContentPane();
14
15         JDesktopPane desktopPane = new JDesktopPane();
16         internalFrame = new JInternalFrame(
17             "Internal Frame", true, //可改变窗口大小
18                 true, //可关闭窗口
19                 true, //可最大化
20                 true); //可最小化
21
22         internalFrame.setLocation(20,20);
23         internalFrame.setSize(200,200);
24         internalFrame.setVisible(true);
25
26         Container icontentPane = internalFrame.getContentPane();
27         JButton b = new JButton("Show Internal Dialog");
28         b.addActionListener(this);
29         icontentPane.add(b, BorderLayout.CENTER);
30         JLabel label = new JLabel(" ", JLabel.CENTER);
31         icontentPane.add(label, BorderLayout.NORTH);
32
33         desktopPane.add(internalFrame);
34
35         contentPane.add(desktopPane, BorderLayout.CENTER);
36         f.setSize(350, 350);
37         f.setVisible(true);
38
39         f.addWindowListener(new WindowAdapter() {
40             public void windowClosing(WindowEvent e) {
41                 System.exit(0);
42             }
43         });
44     }
45
46     public static void main(String[] args)
47     {
48         new InternalDialog();
49     }
50
51     public void actionPerformed(ActionEvent e)
52     {
53         String title = "Option Dialog";
54         String message = "您喜欢吃汉堡吗? ";
55         int messageType = JOptionPane.QUESTION_MESSAGE;
56         int optionType = JOptionPane.YES_NO_CANCEL_OPTION;
57         String[] options = {"喜欢", "不喜欢", "取消"};
58
59         int result = JOptionPane.showInternalOptionDialog(
60             internalFrame, message, title, optionType,
61             messageType, null, options, options[1]);
62
63         if (result == JOptionPane.YES_OPTION)
64             label.setText("您选择: 喜欢");
65         if (result == JOptionPane.NO_OPTION)
```

```

66         label.setText("您选择: 不喜欢");
67     if (result == JOptionPane.CANCEL_OPTION)
68         label.setText("您选择: 取消");
69     if (result == JOptionPane.CLOSED_OPTION)
70         label.setText("您没做任何选择, 并关闭了对话框");
71     }
72 }

```

⊕ 说明:

- (1) 程序第 15~35 行, 使用 Internal Frame, 在此 Internal Frame 中加入一个按钮与一个 JLabel, 并处理 Internal Frame 所产生的按钮事件。
- (2) 默认的 messageType 是 JOptionPane.QUESTION_MESSAGE。
- (3) 程序第 59~61 行, 利用 JOptionPane 的 showInternalOptionDialog() 方法输出 Internal Dialog。当用户在确认对话框中按下按钮或关闭对话框时, 会返回一个整数值来判断用户到底做了什么操作。

程序运行结果如图 11-26 所示。



图 11-26

当按下 “Show Internal Dialog” 按钮时, 如图 11-27 所示。



图 11-27

读者在使用 Internal Dialog 时要特别注意，一般我们利用 JOptionPane 所产生的对话框都是 modal 为 true 状态，可是当您使用 Internal Dialog 时对话框会变成 modal 为 false 状态。因此您可以不用关闭之前的对话框，就可以再按一次按钮，再产生一个 Internal Dialog，如图 11-28 所示。



图 11-28

若您想将 JFrame 内的 Internal Dialog model 设为 true，你可以用下面两种方式解决：

1. 您可以建立 JOptionPane 对象，而不用直接去调用 JOptionPane 的 static 方法来输出对话框。然后利用 JOptionPane 的 createDialog() 方法，取得 JDialog 对象，再利用 Dialog (JDialog 继承 Dialog) 所提供的 setModal() 方法将 modal 设为 true。
2. 直接使用 JDialog。

11-3 使用 JOptionPane 组件建立对话框

我们在 11-2 节中已经介绍了 JOptionPane 的构造函数，虽然大部分的情况下我们只需要使用 JOptionPane 的静态方法来产生对话框，但如果您想直接使用 JOptionPane 对象来产生对话框，当然也可以。下面我们就举一个以 JOptionPane 对象来产生对话框的范例：

范例 OptionPaneDemo.java (文件位于随书光盘目录 exam\ch11\OptionPaneDemo.java)

```
1  import java.awt.*;
2  import javax.swing.*;
3  import java.awt.event.*;
4
5  public class OptionPaneDemo implements ActionListener
6  {
7      JFrame f = null;
8      JLabel label = null;
9
10     public OptionPaneDemo()
11     {
12         f = new JFrame("OptionPane Demo");
13         Container contentPane = f.getContentPane();
14
15         JButton b = new JButton("Show Text Input");
16         b.addActionListener(this);
```

```

17
18     label = new JLabel(" ", JLabel.CENTER);
19     contentPane.add(label, BorderLayout.NORTH);
20     contentPane.add(b, BorderLayout.CENTER);
21     f.pack();
22     f.setVisible(true);
23
24     f.addWindowListener(new WindowAdapter() {
25         public void windowClosing(WindowEvent e) {
26             System.exit(0);
27         }
28     });
29 }
30
31 public static void main(String[] args)
32 {
33     new OptionPaneDemo();
34 }
35
36 public void actionPerformed(ActionEvent e)
37 {
38     String title = "Input Dialog";
39     JLabel message = new JLabel("您最喜欢吃什么食物? ",
40     JLabel.CENTER);
41     int messageType = JOptionPane.QUESTION_MESSAGE;
42     int optionType = JOptionPane.OK_CANCEL_OPTION;
43     String result = "";
44
45     JOptionPane optionPane = new
46     JOptionPane(message, messageType, optionType);
47     optionPane.setWantsInput(true);
48     optionPane.setInitialSelectionValue("请输入! ");
49     optionPane.setInputValue("您没有输入! ");
50     JDialog dialog = optionPane.createDialog(f, title);
51     dialog.show();
52
53     result = (String)optionPane.getInputValue();
54
55     label.setText("您输入: "+result);
56 }
57 }
58

```

◆ 说明:

- (1) 程序第 38~46 行, 利用 message、messageType、optionType 来建立 JOptionPane 对象。
- (2) 程序第 47 行, 利用 JOptionPane 的 setWantsInput() 方法, 使对话框有一个输入字段让用户输入信息。
- (3) 程序第 48 行, JOptionPane 的 setInitialSelectionValue() 方法, 使得输入字段上的初始值为“请输入!”。
- (4) 程序第 49 行, JOptionPane 的 setInputValue() 方法, 使得当用户按下“撤消”按钮或关闭对话框时, result 的默认字符串为“您没有输入!”。

(5) 程序第 53 行, `JOptionPane` 的 `getInputValue()` 方法, 可以取得用户输入的信息。

④ 程序运行结果如图 11-29 所示。



图 11-29

当用户按下“Show Text Input”按钮时, 如图 11-30 所示。



图 11-30

当您输入文字“日本料理”后按下“确定”按钮时, 如图 11-31 所示。



图 11-31

11-4 本章总结

善用对话框可以增进系统与用户之间的沟通, 并在必要时提供给用户相关信息, 帮助用户了解系统状况。在本章中, 我们详细地介绍了如何使用 `JDialog` 与 `JOptionPane`, 并说明了它们之间的差异与使用时机。下次若您设计一套系统让用户输入信息时, 用户若没按照规格填写表格, 别忘了给他一个提示的对话框。

11-5 本章习题

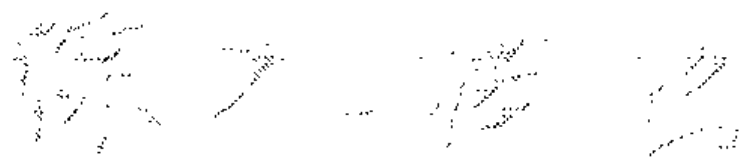
1. 试说明 `JDialog` 与 `JOptionPane` 的差异。
2. 试说明利用 `JOptionPane` 的静态方法可产生几种对话框, 它们的差别是什么?
3. 试改写 `DialogDemo.java` 程序, 使得成为比较完整的“物品借用系统”。
4. 试改写 `InternalDialog.java` 程序, 使得对话框的 `modal` 为 `true` 状态。
5. 试写一个简单的程序, 建立 `JOptionPane` 对象使得 `Input Dialog` 中具有 `ComboBox` 组件, 并取得用户点选选项。

12

菜单与工具栏的使用与介绍

(Menu、Menu Item、Check Box Menu Item、Radio Button Menu Item、Tool Bar、Tool Tips、Popup Menu)

菜单和工具栏已经是软件中必备的组件之一了，菜单和工具栏可提供简单明了的指示说明，让用户更顺利地完成软件的操作。因此，了解并建立一个目录菜单和工具栏是相当重要的工作，我们在本章中会针对目录菜单和工具栏相关组件做详细地介绍。



12-1 使用 JMenuBar 组件

JMenuBar 的类层次结构图:

```
java.lang.Object
--java.awt.Component
--java.awt.Container
--javax.swing.JComponent
--javax.swing.JMenuBar
```

在介绍 JMenu 组件前,我们先介绍 JMenuBar 组件,JMenuBar 组件的功能是用来摆放 JMenu 组件。当我们建立完许多的 JMenu 组件后,需要通过 JMenuBar 组件来将 JMenu 组件加入到窗口中。虽然我们由下表中看出 JMenuBar 组件只有一种构造方式,但是它对于构造一个菜单来说是个不可缺少的组件,如表 12-1 所示。

表 12-1

JMenuBar 的构造函数
JMenuBar() 建立一个新的 JMenuBar

由于构造一个空的 JMenuBar 然后设置到窗口上对于窗口来说是没有意义的,因此 JMenuBar 需要结合至少一个以上的 JMenu 组件才会在画面上显现出视觉效果。所以 JMenuBar 的构造方法及范例我们留到 JMenu 的第一个范例中再加以说明。

12-2 使用 JMenu 组件

JMenu 的类层次结构图:

```
java.lang.Object
--java.awt.Component
--java.awt.Container
--javax.swing.JComponent
--javax.swing.AbstractButton
--javax.swing.JMenuItem
--javax.swing.JMenu
```

JMenu 组件是用来存放和整合 JMenuItem 的组件,这个组件也是构成一个菜单不可或缺的组件之一。JMenu 可以是单一层次的结构也可以是一个层次式的结构,要使用何种形式的结构取决于界面设计上的需要而定,如表 12-2 所示。

表 12-2

JMenu 的构造函数
JMenu() 建立一个新的 JMenu
JMenu(Action a) 建立一个支持 Action 的新的 JMenu
JMenu(String s) 以指定的字符串名称建立一个新的 JMenu

续上表

JMenu 的构造函数

JMenu(String s, Boolean b)

以指定的字符串名称建立一个新的 JMenu 并决定这个菜单是否具有下拉式的属性

12-2-1 构造 JMenu 组件

在看过 JMenu 的构造函数之后，我们先来看一个具有图标菜单的范例：

范例 JMenu1.java (文件位于随书光盘目录 exam\ch12\JMenu1.java)

```

1  import javax.swing.*;
2  import java.awt.event.*;
3
4  public class JMenu1 extends JFrame{
5
6      JTextArea theArea = null;
7
8      public JMenu1(){
9
10         super("JMenu1");
11         theArea = new JTextArea();
12         theArea.setEditable(false);
13         getContentPane().add(new JScrollPane(theArea));
14         JMenuBar MBar = new JMenuBar();
15
16         JMenu mfile = buildFileMenu();
17
18         MBar.add(mfile);
19         setJMenuBar(MBar);
20     }
21
22     public JMenu buildFileMenu() {
23
24         JMenu thefile = new JMenu("File");
25         thefile.setIcon(new ImageIcon("icons/file.gif"));
26
27         return thefile;
28     }
29
30     public static void main(String[] args){
31
32         JFrame F = new JMenu1();
33         F.setSize(400,200);
34         F.addWindowListener(new WindowAdapter() {
35             public void windowClosing(WindowEvent e) {
36                 System.exit(0);
37             }
38         });
39         F.setVisible(true);
40     }
41 }

```


⊕ 说明:

- (1) 程序第 6 行, 初始化一个空的 JTextArea 组件。
- (2) 程序第 11 行, 构造一个新的 JTextArea, 并且在程序第 12 行将 JTextArea 设置为不可编辑。
- (3) 程序第 13 行, 将 JTextArea 加入滚动条列并放入窗口中。
- (4) 程序第 14 行, 构造一个新的 JMenuBar。
- (5) 程序第 16 行, 调用自行编写的 buildFileMenu()方法来构造 JMenu。构造 JMenu 的部分在程序第 22~28 行, 我们利用 setIcon()方法加入图标到 JMenu 中。
- (6) 程序第 18 行, 将 JMenu 加入 JMenuBar 中。
- (7) 程序第 19 行, 将 JMenuBar 设置到窗口中。

⊕ 程序运行结果如图 12-1 所示。



图 12-1

在上面这个例子中我们使用默认字符串的方式来构造 JMenu, 这样的构造方式是最常用的构造方式。我们当然也可以使用其他方式构造来达到相同的效果, 如把程序第 24 行改为下列的方式编写:

```
JMenu thefile = new JMenu();  
thefile.setText("File");
```

将会得到与程序第 25 行一模一样的效果。这样的编写方式是先构造一个空的 JMenu 组件再将 JMenu 的文字设置上去。至于其他两种 JMenu 的构造方式虽然有构造函数可以构造 JMenu 但是其内部相关的属性设置都尚未被 Swing 实现, 因此即使利用该构造函数来构造 JMenu 也只会具备最基本的功能和属性。这个问题必须等到 Swing 推出新版本后才有可能改善, 所以我们就不对另外两种构造方式多做说明了。

12-3 使用 JMenuitem 组件

JMenuItem 的类层次结构图:

```
java.lang.Object  
--java.awt.Component  
--java.awt.Container  
--javax.swing.JComponent  
--javax.swing.AbstractButton  
--javax.swing.JMenuItem
```

第12章 菜单与工具栏的使用与介绍

JMenuItem 继承了 AbstractButton 类，因此 JMenuItem 具有许多 AbstractButton 的特性，也可以说 JMenuItem 是一种特殊的 Button。所以 JMenuItem 支持了许多在 Button 中好用的功能，例如加入图标文件 (Icon) 或是当我们在菜单中选择到某一项 JMenuItem 时就如同按下按钮的操作一样会触发 ActionEvent，通过 ActionEvent 的机制我们就能针对不同的 JMenuItem 编写与其对应的程序区段。我们马上来看看 JMenuItem 的构造方式有哪些，如表 12-3 所示。

表 12-3

JMenuItem 的构造函数
JMenuItem() 建立一个新的 JMenuItem
JMenuItem(Action a) 建立一个支持 Action 的新的 JMenuItem
JMenuItem(Icon icon) 建立一个有图标的 JMenuItem
JMenuItem(String text) 建立一个有文字的 JMenuItem
JMenuItem(String text, Icon icon) 建立一个有图标和文字的 JMenuItem
JMenuItem(String text, int mnemonic) 建立一个有文字和键盘快捷键的 JMenuItem

12-3-1 构造 JMenuItem 组件

在看过 JMenuItem 的构造方式之后，我们马上来看一个例子了解 JMenuItem 是如何运作的。这个例子延伸自 JMenu1.java，改写 buildFileMenu() 方法的部分：

范例 JMenuItem1.java (文件位于随书光盘目录 exam\ch12\JMenuItem1.java)

```

1  import javax.swing.*;
2  import java.awt.event.*;
3
4  public class JMenuItem1 extends JFrame{
5
6      JTextArea theArea = null;
7
8      public JMenuItem1(){
9
10         super("JMenuItem1");
11         theArea = new JTextArea();
12         theArea.setEditable(false);
13         getContentPane().add(new JScrollPane(theArea));
14         JMenuBar MBar = new JMenuBar();
15         MBar.setOpaque(true);
16
17         JMenu mfile = buildFileMenu();
18
19         MBar.add(mfile);
20         setJMenuBar(MBar);
21     }
22

```

```
23     public JMenu buildFileMenu() {
24
25         JMenu thefile = new JMenu("File");
26
27         JMenuItem newf = new JMenuItem("New");
28         JMenuItem open = new JMenuItem("Open");
29         JMenuItem close = new JMenuItem("Close");
30         JMenuItem quit = new JMenuItem("Exit");
31
32         thefile.add(newf);
33         thefile.add(open);
34         thefile.add(close);
35         thefile.addSeparator();
36         thefile.add(quit);
37
38         return thefile;
39     }
40
41     public static void main(String[] args){
42
43         JFrame F = new JFrame();
44         F.setSize(400,200);
45         F.addWindowListener(new WindowAdapter() {
46             public void windowClosing(WindowEvent e) {
47                 System.exit(0);
48             }
49         });
50         F.setVisible(true);
51     }
52 }
```

⊕ 说明:

- (1) 程序第 15 行, 将 JMenuBar 设为不透明, 此为默认值。若您将它设为透明, 则当您在选择菜单时, 会有残影存留在 JMenuBar 上。
- (2) 程序第 27~30 行, 利用指定字符串的构造函数建立四个 JMenuItem。
- (3) 程序第 32~36 行, 将四个 JMenuItem 加入 Menu 中。其中程序第 35 行利用 addSeparator()方法在 JMenu 中加入一条分隔线。
- (4) 程序第 43 行, 建立一个 JFrame 对象。

⊕ 程序运行结果如图 12-2 所示。



图 12-2

我们在一开始提到过 `JMenuItem` 是一个特殊的 `Button` 组件，因此我们可以在 `JMenuItem` 中加入图标来美化显示界面。我们马上来看一个范例：

范例 `JMenuItem2.java` (文件位于随书光盘目录 `exam\ch12\JMenuItem2.java`)

```

1  import javax.swing.*;
2  import java.awt.event.*;
3
4  public class JMenuItem2 extends JFrame{
5
6      JTextArea theArea = null;
7
8      public JMenuItem2(){
9
10         super("JMenuItem2");
11         theArea = new JTextArea();
12         theArea.setEditable(false);
13         getContentPane().add(new JScrollPane(theArea));
14         JMenuBar MBar = new JMenuBar();
15         MBar.setOpaque(true);
16
17         JMenu mfile = buildFileMenu();
18
19         MBar.add(mfile);
20         setJMenuBar(MBar);
21     }
22
23     public JMenu buildFileMenu() {
24
25         JMenu thefile = new JMenu("File");
26
27         JMenuItem newf = new JMenuItem("New",
28 new ImageIcon("icons/new24.gif"));
29         JMenuItem open = new JMenuItem("Open",
30 new ImageIcon("icons/open24.gif"));
31         JMenuItem close= new JMenuItem("Close",
32 new ImageIcon("icons/close24.gif"));
33         JMenuItem quit = new JMenuItem("Exit",
34 new ImageIcon("icons/exit24.gif"));
35
36         thefile.add(newf);
37         thefile.add(open);
38         thefile.add(close);
39         thefile.addSeparator();
40         thefile.add(quit);
41
42         return thefile;
43     }
44
45     public static void main(String[] args){
46
47         JFrame F = new JMenuItem2();
48         F.setSize(400,200);
49         F.addWindowListener(new WindowAdapter() {
50             public void windowClosing(WindowEvent e) {

```

```

51         System.exit(0);
52     }
53 });
54 F.setVisible(true);
55 }
56 }

```

⊕ 说明:

程序 27~34 行, 将构造 JMenuItem 的方式改变成具有字符串和图标文件初始值的方式, 分别指定各个 JMenuItem 的显示字符串和图标文件。

⊕ 程序运行结果如图 12-3 所示。



图 12-3

我们可以看到在文字的左边出现了我们所指定的图标文件。这时我们又会有疑问了, 图标文件一定要摆在文字的左边吗? 答案是不一定的, 图标在这里会出现在文字左边是因为 JMenuItem 默认值的关系, 我们可以利用 setHorizontalTextPosition() 方法改变文字的位置。我们将 JMenuItem2.java 改写成 JMenuItem2e.java 将各个 JMenuItem 的文字位置设置在左边。

范例 JMenuItem2e.java (文件位于随书光盘目录 exam\ch12\JMenuItem2e.java)

在这段程序中我们只改变了 JMenuItem 的字符串显示位置, 因此在这里我们只列出 buildMenu() 方法的程序部分, 如果需要完整的程序代码请查阅附录的光盘内容。

```

1     public JMenu buildFileMenu() {
2
3         JMenu thefile = new JMenu("File");
4
5         JMenuItem newf = new JMenuItem("New",
6     new ImageIcon("icons/new24.gif"));
7         JMenuItem open = new JMenuItem("Open",
8     new ImageIcon("icons/open24.gif"));
9         JMenuItem close = new JMenuItem("Close",
10    new ImageIcon("icons/close24.gif"));
11        JMenuItem quit = new JMenuItem("Exit",
12    new ImageIcon("icons/exit24.gif"));
13
14        newf.setHorizontalTextPosition(SwingConstants.LEFT);
15        open.setHorizontalTextPosition(SwingConstants.LEFT);
16        close.setHorizontalTextPosition(SwingConstants.LEFT);
17        quit.setHorizontalTextPosition(SwingConstants.LEFT);

```

```

18
19     thefile.add(newf);
20     thefile.add(open);
21     thefile.add(close);
22     thefile.addSeparator();
23     thefile.add(quit);
24
25     return thefile;
26 }

```

说明：

程序第 14~17 行，分别将各个 JMenuItem 的文字位置设置到图标文件的左边。在这里我们使用了 SwingConstants 这个 Interface，这个 Interface 定义了许多在 Swing 中常用的数值，由于 AbstractButton 类实作了 SwingConstants Interface，所以我们可以直接使用 SwingConstants.LEFT 表示 JMenuItem 组件的最左边界值。

程序运行结果如图 12-4 所示。



图 12-4

到目前为止我们已经能将一个菜单的外观构造起来了，不过我们还少了一个小小的功能，那就是快捷键的设置。快捷键是让我们能使用键盘来控制菜单方便操作之用。那么怎么样将快捷键加入菜单呢？我们来看看下面这个范例：

范例 JMenuItem3.java (文件位于图书光盘目录 exam\ch12\JMenuItem3.java)

这部分程序修改自 JMenuItem2.java，由于只修改了 buildMenu() 方法，所以在这里我们只列出 buildMenu() 方法的程序部分。如果需要完整的程序代码请查阅附录的光盘内容。

```

1     public JMenu buildFileMenu() {
2
3         JMenu thefile = new JMenu("File");
4         thefile.setMnemonic('F');
5
6         JMenuItem newf = new JMenuItem("New",
7     new ImageIcon("icons/new24.gif"));
8         JMenuItem open = new JMenuItem("Open",
9     new ImageIcon("icons/open24.gif"));
10        JMenuItem close= new JMenuItem("Close",
11    new ImageIcon("icons/close24.gif"));
12        JMenuItem quit = new JMenuItem("Exit",
13    new ImageIcon("icons/exit24.gif"));
14

```

```

15         newf.setMnemonic('N');
16         open.setMnemonic('O');
17         close.setMnemonic('C');
18         quit.setMnemonic('X');
19
20         thefile.add(newf);
21         thefile.add(open);
22         thefile.add(close);
23         thefile.addSeparator();
24         thefile.add(quit);
25
26         return thefile;
27     }

```

说明：

- (1) 程序第 4 行，利用 `setMnemonic()` 方法来设置 `JMenu` 的快捷键。`JMenu` 的快捷键可以在程序运行后以【Alt+快捷键】方式来运行 `JMenu` 的选项。在这里按组合键【Alt+F】就能获得用鼠标点选“File”选项的效果。
- (2) 程序第 15~18 行，分别设置 `JMenuItem` 的快捷键提示。在 `JMenuItem` 中 `setMnemonic()` 方法只是将某个字符加上底线来提示快捷键的字符，一般您只需要键入这些提示字符，就能够运行相关功能，不再需要【Alt】键的辅助。

程序运行结果如图 12-5 所示。



图 12-5

一般的快捷键会设置为【Ctrl】键加上某个字符，来直接运行某项功能，图 12-6 为 Word 中快捷键的图标：



图 12-6

要在菜单中加入快捷键很简单，只需要使用 `setAccelerator()` 方法即可，我们来看下面的范例：

范例 JMenuitem3e.java (文件位于随书光盘目录 exam\ch12\JMenuitem3e.java)

这部分程序修改自 `JMenuItem3.java`，由于只修改了 `bulidMenu()` 方法，所以在这里我们只列出 `bulidMenu()` 方法的程序部分，如果需要完整的程序代码请查阅附录的光盘内容。

```

1      public JMenu buildFileMenu() {
2
3          JMenu thefile = new JMenu("File");
4          thefile.setMnemonic('F');
5
6          JMenuItem newf = new JMenuItem("New",
7      new ImageIcon("icons/new24.gif"));
8          JMenuItem open = new JMenuItem("Open",
9      new ImageIcon("icons/open24.gif"));
10         JMenuItem close= new JMenuItem("Close",
11     new ImageIcon("icons/close24.gif"));
12         JMenuItem quit = new JMenuItem("Exit",
13     new ImageIcon("icons/exit24.gif"));
14
15         newf.setMnemonic('N');
16         open.setMnemonic('O');
17         close.setMnemonic('C');
18         quit.setMnemonic('X');
19
20         newf.setAccelerator( KeyStroke.getKeyStroke
21     ('N', java.awt.Event.CTRL_MASK, false) );
22         open.setAccelerator( KeyStroke.getKeyStroke
23     ('O', java.awt.Event.CTRL_MASK, false) );
24         close.setAccelerator( KeyStroke.getKeyStroke
25     ('L', java.awt.Event.CTRL_MASK, false) );
26         quit.setAccelerator( KeyStroke.getKeyStroke
27     ('X', java.awt.Event.CTRL_MASK, false) );
28
29         thefile.add(newf);
30         thefile.add(open);
31         thefile.add(close);
32         thefile.addSeparator();
33         thefile.add(quit);
34
35         return thefile;
36     }

```

说明：

程序第 20~27 行，利用 `setAccelerator()` 方法，分别设置各个 `JMenuItem` 的快捷键值。在 `setAccelerator()` 方法中使用了 `KeyStroke` 类，这个类用来管理键盘上的各种信息，我们利用 `getKeyStroke()` 方法来指定快捷键的键值。`getKeyStroke()` 方法的三个字段值分别代表键值、屏蔽键值和放开按键时是否触发事件。在这个范例里我们设置【Ctrl+英文字符】来运行 `JMenuItem` 的选项功能。

④ 程序运行结果如图 12-7 所示。



图 12-7

当然，快捷键的屏蔽键值不一定是【Ctrl】键，我们也可以将屏蔽键值设置为其他的功能键，如【Shift】、【Alt】……。如我们将 JMenuItem3e.java 程序中的第 20~27 行改为下列：

```
newf.setAccelerator( KeyStroke.getKeyStroke
('N', java.awt.Event.SHIFT_MASK, false) );
open.setAccelerator( KeyStroke.getKeyStroke
('O', java.awt.Event.SHIFT_MASK, false) );
lose.setAccelerator( KeyStroke.getKeyStroke
('L', java.awt.Event.SHIFT_MASK, false) );
quit.setAccelerator( KeyStroke.getKeyStroke
('X', java.awt.Event.SHIFT_MASK, false) );
```

另存于程序 JMenuItem3e2.java 中，则 JMenuItem 的快捷键变成【Shift+英文字符】。

④ 程序运行结果如图 12-8 所示。



图 12-8

现在我们已经能够完整的建立一个菜单了，但是我们现在看到的菜单都是单一层次的，如何来建立具有层次式的菜单呢？我们来看下面的范例：

范例 JMenuItem4.java (文件位于随书光盘目录 exam\ch12\JMenuItem4.java)

这部分程序修改自 JMenuItem3.java，由于只修改了 buildMenu() 方法，所以在这里我们只列出 buildMenu() 方法的程序部分，如果需要完整的程序代码请查阅附录的光盘内容。

```
1 public JMenu buildFileMenu() {
2
3     JMenu thefile = new JMenu("File");
```

```

4         thefile.setMnemonic('F');
5
6         JMenuItem newf = new JMenuItem
7 ("New",new ImageIcon("icons/new24.gif"));
8         JMenuItem open = new JMenuItem
9 ("Open",new ImageIcon("icons/open24.gif"));
10        JMenuItem close= new JMenuItem
11 ("Close",new ImageIcon("icons/close24.gif"));
12        JMenuItem quit = new JMenuItem
13 ("Exit",new ImageIcon("icons/exit24.gif"));
14
15        newf.setMnemonic('N');
16        open.setMnemonic('O');
17        close.setMnemonic('L');
18        quit.setMnemonic('X');
19
20        newf.setAccelerator( KeyStroke.getKeyStroke
21 ('N', java.awt.Event.CTRL_MASK, false) );
22        open.setAccelerator( KeyStroke.getKeyStroke
23 ('O', java.awt.Event.CTRL_MASK, false) );
24        close.setAccelerator( KeyStroke.getKeyStroke
25 ('L', java.awt.Event.CTRL_MASK, false) );
26        quit.setAccelerator( KeyStroke.getKeyStroke
27 ('X', java.awt.Event.CTRL_MASK, false) );
28
29        JMenu prefMenu = new JMenu("Preferences..");
30        prefMenu.setMnemonic('P');
31
32        JMenuItem setPage = new JMenuItem
33 ("setPage",new ImageIcon("icons/setpage24.gif"));
34        JMenuItem setImport = new JMenuItem
35 ("Import",new ImageIcon("icons/import24.gif"));
36        JMenuItem setExport = new JMenuItem
37 ("Export",new ImageIcon("icons/export24.gif"));
38
39        setPage.setMnemonic('S');
40        setImport.setMnemonic('I');
41        setExport.setMnemonic('E');
42
43        setPage.setAccelerator( KeyStroke.getKeyStroke
44 ('S', java.awt.Event.CTRL_MASK, false) );
45        setImport.setAccelerator( KeyStroke.getKeyStroke
46 ('I', java.awt.Event.CTRL_MASK, false) );
47        setExport.setAccelerator( KeyStroke.getKeyStroke
48 ('E', java.awt.Event.CTRL_MASK, false) );
49
50        prefMenu.add(setPage);
51        prefMenu.add(setImport);
52        prefMenu.add(setExport);
53
54        thefile.add(newf);
55        thefile.add(open);
56        thefile.add(close);
57        thefile.addSeparator();
58        thefile.add(prefMenu);

```

```

59     thefile.addSeparator();
60     thefile.add(quit);
61
62     return thefile;
63 }

```

说明：

- (1) 程序第 29 行，构造一个名为“Preferences..”的 JMenu 组件。
- (2) 程序第 30 行，设置这个 JMenu 的快捷键为“P”。
- (3) 程序第 32~37 行，构造三个 JMenuItem 组件准备放到 JMenu 中。
- (4) 程序第 39~41 行，设置三个 JMenuItem 的快捷键提示。
- (5) 程序第 43~48 行，分别指定三个 JMenuItem 的快捷键。
- (6) 程序第 50~52 行，将三个 JMenuItem 加入 JMenu 中。
- (7) 程序第 58 行，将 JMenu 组件加入 JMenu 组件中达到层次式 Menu 的效果。

程序运行结果如图 12-9 所示。



图 12-9

12-3-2 JMenuItem 的事件处理

由于 JMenuItem 继承了 AbstractButton 类，因此 JMenuItem 也具备了许多 AbstractButton 的特性，当然也包含了事件处理的机制。JMenuItem 的事件处理机制类似于 JButton 的事件处理机制，换句话说，当按下 JMenuItem 组件时就如同按下 JButton 组件一般，均会产生(ActionEvent)事件。我们马上来看一个范例：

范例 JMenuItem5.java (文件位于随书光盘目录 exam\ch12\JMenuItem5.java)

```

1  import javax.swing.*;
2  import java.awt.event.*;
3
4  public class JMenuItem5 extends JFrame{
5
6      JTextArea theArea = null;
7
8      public JMenuItem5(){

```

```

9
10     super("JMenuItem5");
11     theArea = new JTextArea();
12     theArea.setEditable(false);
13     getContentPane().add(new JScrollPane(theArea));
14     JMenuBar MBar = new JMenuBar();
15     MBar.setOpaque(true);
16
17     JMenu mfile = buildFileMenu();
18
19     MBar.add(mfile);
20     setJMenuBar(MBar);
21 }
22
23 public JMenu buildFileMenu() {
24
25     JMenu thefile = new JMenu("File");
26     thefile.setMnemonic('F');
27
28     JMenuItem newf = new JMenuItem
29 ("New",new ImageIcon("icons/new24.gif"));
30     JMenuItem open = new JMenuItem
31 ("Open",new ImageIcon("icons/open24.gif"));
32     JMenuItem close= new JMenuItem
33 ("Close",new ImageIcon("icons/close24.gif"));
34     JMenuItem quit = new JMenuItem
35 ("Exit",new ImageIcon("icons/exit24.gif"));
36
37     newf.setMnemonic('N');
38     open.setMnemonic('O');
39     close.setMnemonic('L');
40     quit.setMnemonic('X');
41
42     newf.setAccelerator( KeyStroke.getKeyStroke
43 ('N', java.awt.Event.CTRL_MASK, false) );
44     open.setAccelerator( KeyStroke.getKeyStroke
45 ('O', java.awt.Event.CTRL_MASK, false) );
46     close.setAccelerator( KeyStroke.getKeyStroke
47 ('L', java.awt.Event.CTRL_MASK, false) );
48     quit.setAccelerator( KeyStroke.getKeyStroke
49 ('X', java.awt.Event.CTRL_MASK, false) );
50
51     newf.addActionListener(new ActionListener() {
52         public void actionPerformed(ActionEvent e) {
53             theArea.append("- MenuItem New Performed -\n");
54         }
55     });
56
57     open.addActionListener(new ActionListener() {
58         public void actionPerformed(ActionEvent e) {
59             theArea.append("- MenuItem Open Performed -\n");
60         }
61     });
62
63     close.addActionListener(new ActionListener() {
64         public void actionPerformed(ActionEvent e) {
65             theArea.append("- MenuItem Close Performed -\n");
66         }
67     });
68 }

```

```
64         });
65
66         quit.addActionListener(new ActionListener() {
67             public void actionPerformed(ActionEvent e) {
68                 System.exit(0);
69             }
70
71         thefile.add(newf);
72         thefile.add(open);
73         thefile.add(close);
74         thefile.addSeparator();
75         thefile.add(quit);
76
77         return thefile;
78     }
79
80     public static void main(String[] args){
81
82         JFrame F = new JMenuItem5();
83         F.setSize(400,200);
84         F.addWindowListener(new WindowAdapter() {
85             public void windowClosing(WindowEvent e) {
86                 System.exit(0);
87             }
88         });
89         F.setVisible(true);
90     }
91 }
```

◆ 说明

- (1) 本程序延伸自 JMenuItem3.java，加入了 JMenuItem 的事件处理模式，以匿名类的方式来编写相对应事件的操作。
- (2) 程序第 51 行，利用 addActionListener() 方法将 JMenuItem 加入事件处理模式中。程序第 52~54 行，采用匿名类的方式编写当被事件触发时产生操作的部分。在这里是将一个字符串加入 JTextArea 中。
- (3) 在程序第 56~59 行、程序第 61~64 行、程序第 66~69 行分别将另外三个 JMenuItem 加入事件处理模式。

◆ 程序运行结果如图 12-10 所示。

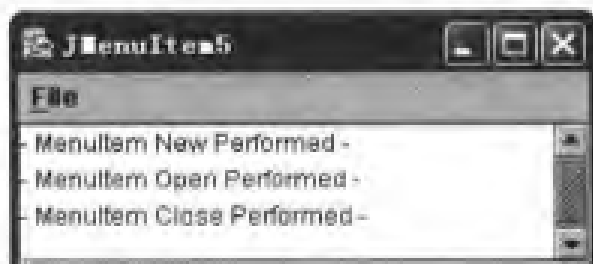


图 12-10

12-4 使用 JCheckBoxMenuItem

JCheckBoxMenuItem 的类层次结构图:

```
java.lang.Object
--java.awt.Component
--java.awt.Container
--javax.swing.JComponent
--javax.swing.AbstractButton
--javax.swing.JMenuItem
--javax.swing.JCheckBoxMenuItem
```

JCheckBoxMenuItem 继承了 JMenuItem 类, 因此 JCheckBoxMenuItem 可以使用 JMenuItem 所提供的方法, 而且 JCheckBoxMenuItem 也具有 AbstractButton 的特性。JCheckBoxMenuItem 可以说是一种特殊的 JMenuItem。我们在前面有介绍过 JCheckBox 组件, 而 JCheckBoxMenuItem 和 JCheckBox 的性质几乎是一样的, 两者间最大的差别在于 JCheckBoxMenuItem 是专用在 MenuItem 上。我们来看看 JCheckBoxMenuItem 的构造方法, 如表 12-4 所示。

表 12-4

JCheckBoxMenuItem 的构造函数	
JCheckBoxMenuItem()	建立一个新的 JCheckBoxMenuItem
JCheckBoxMenuItem(Action a)	建立一个支持 Action 的新的 JCheckBoxMenuItem
JCheckBoxMenuItem(Icon icon)	建立一个有图标的 JCheckBoxMenuItem
JCheckBoxMenuItem(String text)	建立一个有文字的 JCheckBoxMenuItem
JCheckBoxMenuItem(String text, Boolean b)	建立一个有文字和设置选择状态的 JCheckBoxMenuItem
JCheckBoxMenuItem(String text, Icon icon)	建立一个有文字和图标的 JCheckBoxMenuItem
JCheckBoxMenuItem(String text, Icon icon, Boolean b)	建立一个有文字、图标和设置选择状态的 JCheckBoxMenuItem

12-4-1 构造 JCheckBoxMenuItem 组件

由上表可以看出 JCheckBoxMenuItem 的构造方式与 JMenuItem 的构造方式几乎一模一样, 唯一有差异的地方是 JCheckBoxMenuItem 的构造方式中多了设置选择状况的构造方式, 设置选择状态就是决定是否要将构造好的 JCheckBoxMenuItem 组件设置成默认值。我们马上来看一个范例:

范例 JCheckBoxMenuItem1.java (文件位于随书光盘目录 exam\ch12\JcheckboxBoxMenuItem1.java)

```
1 import javax.swing.*;
2 import java.awt.event.*;
3
```

```
4 public class JCheckBoxMenuItem1 extends JFrame
5 implements ActionListener{
6
7     JTextArea theArea = null;
8
9     public JCheckBoxMenuItem1(){
10
11         super("JCheckBoxMenuItem1");
12         theArea = new JTextArea();
13         theArea.setEditable(false);
14         getContentPane().add(new JScrollPane(theArea));
15         JMenuBar MBar = new JMenuBar();
16         MBar.setOpaque(true);
17
18         JMenu mfile = buildFileMenu();
19         JMenu mstyle = buildStyleMenu();
20
21         MBar.add(mfile);
22         MBar.add(mstyle);
23         setJMenuBar(MBar);
24     }
25
26     public JMenu buildFileMenu() {
27
28         JMenu thefile = new JMenu("File");
29         thefile.setMnemonic('F');
30
31         JMenuItem newf = new JMenuItem("New",
32 new ImageIcon("icons/new24.gif"));
33         JMenuItem open = new JMenuItem("Open",
34 new ImageIcon("icons/open24.gif"));
35         JMenuItem close= new JMenuItem("Close",
36 new ImageIcon("icons/close24.gif"));
37         JMenuItem quit = new JMenuItem("Exit",
38 new ImageIcon("icons/exit24.gif"));
39
40         newf.setMnemonic('N');
41         open.setMnemonic('O');
42         close.setMnemonic('L');
43         quit.setMnemonic('X');
44
45         newf.setAccelerator( KeyStroke.getKeyStroke
46 ('N', java.awt.Event.CTRL_MASK, false) );
47         open.setAccelerator( KeyStroke.getKeyStroke
48 ('O', java.awt.Event.CTRL_MASK, false) );
49         close.setAccelerator( KeyStroke.getKeyStroke
50 ('L', java.awt.Event.CTRL_MASK, false) );
51         quit.setAccelerator( KeyStroke.getKeyStroke
52 ('X', java.awt.Event.CTRL_MASK, false) );
53
54         newf.addActionListener(this);
55         open.addActionListener(this);
56         close.addActionListener(this);
57         quit.addActionListener(new ActionListener() {
58             public void actionPerformed(ActionEvent e) {
```

```

59             System.exit(0);
60         });
61
62         thefile.add(newf);
63         thefile.add(open);
64         thefile.add(close);
65         thefile.addSeparator();
66         thefile.add(quit);
67
68         return thefile;
69     }
70
71     public JMenu buildStyleMenu() {
72
73         JMenu style = new JMenu("Style");
74         style.setMnemonic('S');
75
76         JCheckBoxMenuItem Left    = new JCheckBoxMenuItem("Left",
77 new ImageIcon("icons/left24.gif"));
78         JCheckBoxMenuItem Center  = new JCheckBoxMenuItem("Center",
79 new ImageIcon("icons/center24.gif"));
80         JCheckBoxMenuItem Right   = new JCheckBoxMenuItem("Right",
81 new ImageIcon("icons/right24.gif"));
82         JCheckBoxMenuItem Justify = new JCheckBoxMenuItem("Justify",
83 new ImageIcon("icons/justify24.gif"));
84
85         Left.setMnemonic('L');
86         Center.setMnemonic('E');
87         Right.setMnemonic('R');
88         Justify.setMnemonic('J');
89
90         Left.setAccelerator( KeyStroke.getKeyStroke
91 ('L', java.awt.Event.SHIFT_MASK, false) );
92         Center.setAccelerator( KeyStroke.getKeyStroke
93 ('E', java.awt.Event.SHIFT_MASK, false) );
94         Right.setAccelerator( KeyStroke.getKeyStroke
95 ('R', java.awt.Event.SHIFT_MASK, false) );
96         Justify.setAccelerator( KeyStroke.getKeyStroke
97 ('J', java.awt.Event.SHIFT_MASK, false) );
98
99         Left.addActionListener(this);
100        Center.addActionListener(this);
101        Right.addActionListener(this);
102        Justify.addActionListener(this);
103
104        style.add(Left);
105        style.add(Center);
106        style.add(Right);
107        style.add(Justify);
108
109        return style;
110    }
111
112    public void actionPerformed(ActionEvent ae){
113        try{

```



```

114         theArea.append(" action '"+ae.getActionCommand()+"
115         " performed. *\n");
116         }catch(Exception e){
117             System.out.println("actionPerformed Exception:"+e);
118         }
119     }
120
121     public static void main(String[] args){
122
123         JFrame F = new JCheckBoxMenuItem();
124         F.setSize(400,200);
125         F.addWindowListener(new WindowAdapter() {
126             public void windowClosing(WindowEvent e) {
127                 System.exit(0);
128             }
129         });
130         F.setVisible(true);
131     }

```

说明：

- (1) 程序第 4~5 行，由于我们要处理 ActionEvent 事件，因此要实现 ActionListener 界面。
- (2) 程序第 19 行，调用 buildStyleMenu()方法来建立包含 JCheckBoxMenuItem 的 JMenu。
- (3) 程序第 54~56 行，将 File Menu 内的三个 JMenuItem 对象加入事件处理模式。
- (4) 程序第 57~60 行，使用匿名类写法处理 File Menu 内第四个 JMenuItem 的事件处理模式。
- (5) 程序第 73 行，建立新的 JMenu 组件并在程序第 74 行设置其快捷键。
- (6) 程序第 76~83 行，分别建立四个 JCheckBoxMenuItem 组件，以默认字符串和图标文件的方式来构造。
- (7) 程序第 85~88 行，分别设置四个 JCheckBoxMenuItem 的快捷键提示。
- (8) 程序第 90~97 行，分别设置四个 JCheckBoxMenuItem 的快捷键。
- (9) 程序第 99~102 行，分别将四个 JCheckBoxMenuItem 加入事件处理模式。
- (10) 程序第 104~107 行，分别将四个 JCheckBoxMenuItem 加入 Menu 菜单中。
- (11) 程序第 112~119 行，为事件处理的区段。在这里是将被点选的 JMenuItem 组件名称填入 JTextArea 中。

程序运行结果如图 12-11 所示。



图 12-11

12-5 使用 JRadioButtonMenuItem 组件

JRadioButtonMenuItem 的类层次结构图:

```
java.lang.Object
--java.awt.Component
--java.awt.Container
--javax.swing.JComponent
--javax.swing.AbstractButton
--javax.swing.JMenuItem
--javax.swing.JRadioButtonMenuItem
```

与 JCheckBoxMenuItem 相同, JRadioButtonMenuItem 也是继承自 JMenuItem, 因此 JRadioButtonMenuItem 也具备 JMenuItem 的许多特性。而 JRadioButtonMenuItem 和 JCheckBoxMenuItem 一样也是一种特殊的 JMenuItem。我们在前面也介绍过 JRadioButton 组件, 而 JRadioButtonMenuItem 和 JRadioButton 的性质几乎完全相同, 两者间最大的差别在于 JRadioButtonMenuItem 是专用在 MenuItem 上。我们来看看 JRadioButtonMenuItem 的构造方法, 如表 12-5 所示。

表 12-5

JRadioButtonMenuItem 的构造函数

JRadioButtonMenuItem()	建立一个新的 JRadioButtonMenuItem
JRadioButtonMenuItem(Action a)	建立一个支持 Action 的新的 JRadioButtonMenuItem
JRadioButtonMenuItem(Icon icon)	建立一个有图标的 JRadioButtonMenuItem
JRadioButtonMenuItem(Icon icon, Boolean selected)	建立一个有图标和设置选择状态的 JRadioButtonMenuItem
JRadioButtonMenuItem(String text)	建立一个有文字的 JRadioButtonMenuItem
JRadioButtonMenuItem(String text, Boolean selected)	建立一个有文字和设置选择状态的 JRadioButtonMenuItem
JRadioButtonMenuItem(String text, Icon icon)	建立一个有文字和图标的 JRadioButtonMenuItem
JRadioButtonMenuItem(String text, Icon icon, Boolean selected)	建立一个有文字、图标和设置选择状态的 JRadioButtonMenuItem

12-5-1 构造 JRadioButtonMenuItem 组件

由上表可以看出 JRadioButtonMenuItem 的构造方式与 JCheckBoxMenuItem 的构造方式几乎完全相同, 而且和 JCheckBoxMenuItem 的构造方式一样都是比 JMenuItem 的构造方式多了设置选择状况的构造方式; 设置选择状态就是决定是否要将构造好的 JRadioButtonMenuItem 设置为默认值。我们马上来看一个范例:

范例 JRadioButtonMenuItem1.java (文件位于随书光盘目录 exam\ch12\JRadioButtonMenuItem1.java)

```

1  import javax.swing.*;
2  import java.awt.event.*;
3
4  public class JCheckBoxMenuItem1 extends JFrame
5  implements ActionListener{
6
7      JTextArea theArea = null;
8
9      public JCheckBoxMenuItem1(){
10
11         super("JCheckBoxMenuItem1");
12         theArea = new JTextArea();
13         theArea.setEditable(false);
14         getContentPane().add(new JScrollPane(theArea));
15         JMenuBar MBar = new JMenuBar();
16         MBar.setOpaque(true);
17
18         JMenu mfile = buildFileMenu();
19         JMenu mstyle = buildStyleMenu();
20
21         MBar.add(mfile);
22         MBar.add(mstyle);
23         setJMenuBar(MBar);
24     }
25
26     public JMenu buildFileMenu() {
27
28         JMenu thefile = new JMenu("File");
29         thefile.setMnemonic('F');
30
31         JMenuItem newf = new JMenuItem("New",
32         new ImageIcon("icons/new24.gif"));
33         JMenuItem open = new JMenuItem("Open",
34         new ImageIcon("icons/open24.gif"));
35         JMenuItem close= new JMenuItem("Close",
36         new ImageIcon("icons/close24.gif"));
37         JMenuItem quit = new JMenuItem("Exit",
38         new ImageIcon("icons/exit24.gif"));
39
40         newf.setMnemonic('N');
41         open.setMnemonic('O');
42         close.setMnemonic('L');
43         quit.setMnemonic('X');
44
45         newf.setAccelerator( KeyStroke.getKeyStroke
46         ('N', java.awt.Event.CTRL_MASK, false) );
47         open.setAccelerator( KeyStroke.getKeyStroke
48         ('O', java.awt.Event.CTRL_MASK, false) );
49         close.setAccelerator( KeyStroke.getKeyStroke
50         ('L', java.awt.Event.CTRL_MASK, false) );
51         quit.setAccelerator( KeyStroke.getKeyStroke
52         ('X', java.awt.Event.CTRL_MASK, false) );

```

```

53
54     newf.addActionListener(this);
55     open.addActionListener(this);
56     close.addActionListener(this);
57     quit.addActionListener(new ActionListener() {
58         public void actionPerformed(ActionEvent e) {
59             System.exit(0);
60         }
61     });
62     thefile.add(newf);
63     thefile.add(open);
64     thefile.add(close);
65     thefile.addSeparator();
66     thefile.add(quit);
67
68     return thefile;
69 }
70
71 public JMenu buildStyleMenu() {
72
73     JMenu style = new JMenu("Style");
74     style.setMnemonic('S');
75
76     JRadioButtonMenuItem Left = new JRadioButtonMenuItem
77     ("Left",new ImageIcon("icons/left24.gif"));
78     JRadioButtonMenuItem Center = new JRadioButtonMenuItem ("Center",
79     new ImageIcon("icons/center24.gif"));
80     JRadioButtonMenuItem Right = new JRadioButtonMenuItem ("Right",
81     new ImageIcon("icons/right24.gif"));
82     JRadioButtonMenuItem Justify = new JRadioButtonMenuItem ("Justify",
83     new ImageIcon("icons/justify24.gif"));
84
85     Left.setMnemonic('L');
86     Center.setMnemonic('E');
87     Right.setMnemonic('R');
88     Justify.setMnemonic('J');
89
90     Left.setAccelerator( KeyStroke.getKeyStroke
91     ('L', java.awt.Event.SHIFT_MASK, false) );
92     Center.setAccelerator( KeyStroke.getKeyStroke
93     ('E', java.awt.Event.SHIFT_MASK, false) );
94     Right.setAccelerator( KeyStroke.getKeyStroke
95     ('R', java.awt.Event.SHIFT_MASK, false) );
96     Justify.setAccelerator( KeyStroke.getKeyStroke
97     ('J', java.awt.Event.SHIFT_MASK, false) );
98
99     Left.addActionListener(this);
100    Center.addActionListener(this);
101    Right.addActionListener(this);
102    Justify.addActionListener(this);
103
104    style.add(Left);
105    style.add(Center);
106    style.add(Right);
107    style.add(Justify);

```

```

108
109     return style;
110 }
111
112 public void actionPerformed(ActionEvent ae){
113     try{
114         theArea.append("* action '"+ae.getActionCommand()+
115 "' performed. *\n");
116     }catch(Exception e){
117         System.out.println("actionPerformed Exception:"+e);
118     }
119 }
120
121 public static void main(String[] args){
122
123     JFrame F = new JCheckBoxMenuItem();
124     F.setSize(400,200);
125     F.addWindowListener(new WindowAdapter() {
126         public void windowClosing(WindowEvent e) {
127             System.exit(0);
128         }
129     });
130     F.setVisible(true);
131 }
132 }

```

◆ 说明:

- (1) 程序第 4~5 行, 由于本程序是直接采用事件托管模式来管理事件处理, 因此要实现 ActionListener。
- (2) 程序第 19 行, 调用 buildStyleMenu()方法来建立包含 JRadioButtonMenuItem 的 JMenu。
- (3) 程序第 54~56 行, 将 File Menu 内的三个 JMenuItem 对象加入事件处理模式。
- (4) 程序第 57~60 行, 使用匿名类写法处理 File Menu 内第四个 JMenuItem 的事件处理模式。
- (5) 程序第 73 行, 建立新的 JMenu 组件并在程序第 74 行设置其快捷键。
- (6) 程序第 76~83 行, 分别建立四个 JRadioButtonMenuItem 组件, 以默认字符串和图标文件的方式来构造。
- (7) 程序第 85~88 行, 分别设置四个 JRadioButtonMenuItem 的快捷键提示。
- (8) 程序第 90~97 行, 分别设置四个 JRadioButtonMenuItem 的快捷键。
- (9) 程序第 99~102 行, 分别将四个 JRadioButtonMenuItem 加入事件处理模式。
- (10) 程序第 104~107 行, 分别将四个 JRadioButtonMenuItem 加入 Menu 菜单中。
- (11) 程序第 112~119 行, 为事件处理的区段。在这里是将被点选的 JMenuItem 组件名称填入 JTextArea 中。

◆ 程序运行结果如图 12-12 所示。



图 12-12

在这个范例中我们遇到了与 `JRadioButton` 同样的问题，那就是选项的单、复选问题。一般来说 `RadioButton` 组件是用来做单选的情况时使用，而 `CheckBox` 则是多为复选情况时使用。那么我们该怎样解决这个问题呢？还记得我们在介绍 `JRadioButton` 时的解决方式吗？没错！就是要利用 `ButtonGroup` 类来处理。我们在 `JRadioButtonMenuItem.java` 的 `buildStyleMenu()` 方法中加上下列五行代码：

```
ButtonGroup bg = new ButtonGroup();
bg.add(Left);
bg.add(Center);
bg.add(Right);
bg.add(Justify);
```

将程序另存为 `JRadioButtonMenuItem2.java`，编译并运行后的结果如图 12-13 所示。



图 12-13

我们可以看到加入 `ButtonGroup` 后的 `JRadioButtonMenuItem` 变成只能够被单选的菜单了。

12-6 使用 JToolBar 组件

`JToolBar` 的类层次结构图：

```
java.lang.Object
--java.awt.Component
--java.awt.Container
--javax.swing.JComponent
--javax.swing.JToolBar
```

ToolBar 的功能是用来放置各种常用的功能或控制组件, 这个功能在各类软件中都可以看到。一般我们在设计软件时, 会将所有功能依类放置在菜单中 (JMenu), 但当功能数量相当多时, 可能会造成用户进行一个简单的操作就必须繁复的寻找菜单中相关的功能, 这将造成用户操作上的负担。若我们能将一般常用的功能以工具栏方式呈现在菜单下, 让用户很快得到他想要的功能, 不仅增加用户使用软件的意愿, 也加速工作的运行效率。这就是使用 ToolBar 的好处。我们现在来看看 JToolBar 的构造方式, 如表 12-6 所示。

表 12-6

JToolBar 的构造函数	
JToolBar()	建立一个新的 JToolBar, 位置为默认的水平方向
JToolBar(int orientation)	建立一个指定位置的 JToolBar
JToolBar(String name)	建立一个指定名称的 JToolBar
JToolBar(String name, int orientation)	建立一个指定名称和位置的 JToolBar

12-6-1 构造 JToolBar 组件

在使用 JToolBar 时一般都是采用水平方向的位置, 因此我们在构造时多是采用上表中的第一种构造方式来建立 JToolBar, 如果需要改变方向时再用 JToolBar 内的 setOrientation() 方法来改变设置, 或是以鼠标拉动的方式来改变 JToolBar 的位置。我们马上来看一个范例:

范例 JToolBar1.java (文件位于随书光盘目录 example12\JToolBar1.java)

```

1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4
5  public class JToolBar1 extends JFrame{
6
7      JTextArea theArea = null;
8      static final String ComboStr[] = {"Times New Roman","Dialog",
9      "细明体","新细明体","标楷体"};
10
11     public JToolBar1(){
12
13         super("JToolBar1");
14         theArea = new JTextArea();
15         theArea.setEditable(false);
16         getContentPane().add(new JScrollPane(theArea));
17         JMenuBar MBar = new JMenuBar();
18         MBar.setOpaque(true);
19
20         JMenu mfile = buildFileMenu();
21         JToolBar theBar = buildToolBar();
22         this.getContentPane().add(theBar,BorderLayout.NORTH);
23

```

```

24     MBar.add(mfile);
25     setJMenuBar(MBar);
26 }
27
28     public JMenu buildFileMenu() {
29
30         JMenu thefile = new JMenu("File");
31         thefile.setMnemonic('F');
32
33         JMenuItem newf = new JMenuItem("New",
34 new ImageIcon("icons/new24.gif"));
35         JMenuItem open = new JMenuItem("Open",
36 new ImageIcon("icons/open24.gif"));
37         JMenuItem close = new JMenuItem("Close",
38 new ImageIcon("icons/close24.gif"));
39         JMenuItem quit = new JMenuItem("Exit",
40 new ImageIcon("icons/exit24.gif"));
41
42         newf.setMnemonic('N');
43         open.setMnemonic('O');
44         close.setMnemonic('L');
45         quit.setMnemonic('X');
46
47         newf.setAccelerator( KeyStroke.getKeyStroke('N',
48 java.awt.Event.CTRL_MASK, false) );
49         open.setAccelerator( KeyStroke.getKeyStroke('O',
50 java.awt.Event.CTRL_MASK, false) );
51         close.setAccelerator( KeyStroke.getKeyStroke('L',
52 java.awt.Event.CTRL_MASK, false) );
53         quit.setAccelerator( KeyStroke.getKeyStroke('X',
54 java.awt.Event.CTRL_MASK, false) );
55
56         newf.addActionListener(new ActionListener() {
57             public void actionPerformed(ActionEvent e) {
58                 theArea.append("- MenuItem New Performed -\n");
59             }
60         });
61
62         open.addActionListener(new ActionListener() {
63             public void actionPerformed(ActionEvent e) {
64                 theArea.append("- MenuItem Open Performed -\n");
65             }
66         });
67
68         close.addActionListener(new ActionListener() {
69             public void actionPerformed(ActionEvent e) {
70                 theArea.append("- MenuItem Close Performed -\n");
71             }
72         });
73
74         quit.addActionListener(new ActionListener() {
75             public void actionPerformed(ActionEvent e) {
76                 System.exit(0);
77             }
78         });
79
80         thefile.add(newf);
81         thefile.add(open);
82         thefile.add(close);
83     }

```



```
79         thefile.addSeparator();
80         thefile.add(quit);
81
82         return thefile;
83     }
84
85     public JToolBar buildToolBar() {
86
87         JToolBar toolBar = new JToolBar();
88         toolBar.setFloatable(true);
89
90         ToolBarAction tba_new = new ToolBarAction("new",
91             new ImageIcon("icons/new24.gif"));
92         ToolBarAction tba_open = new ToolBarAction("open",
93             new ImageIcon("icons/open24.gif"));
94         ToolBarAction tba_close = new ToolBarAction("close",
95             new ImageIcon("icons/close24.gif"));
96
97         JButton JB;
98         JB = toolBar.add(tba_new);
99         JB.setActionCommand("#ToolBar_NEW performed!");
100        JB = toolBar.add(tba_open);
101        JB.setActionCommand("#ToolBar_OPEN performed!");
102        JB = toolBar.add(tba_close);
103        JB.setActionCommand("#ToolBar_CLOSE performed!");
104
105        toolBar.addSeparator();
106
107        ToolBarAction tba_B = new ToolBarAction("bold",
108            new ImageIcon("icons/bold24.gif"));
109        ToolBarAction tba_I = new ToolBarAction("italic",
110            new ImageIcon("icons/italic24.gif"));
111        ToolBarAction tba_U = new ToolBarAction("underline",
112            new ImageIcon("icons/underline24.gif"));
113        JB = toolBar.add(tba_B);
114        JB.setActionCommand("#ToolBar_Bold performed!");
115        JB = toolBar.add(tba_I);
116        JB.setActionCommand("#ToolBar_Italic performed!");
117        JB = toolBar.add(tba_U);
118        JB.setActionCommand("#ToolBar_Underline performed!");
119
120        toolBar.addSeparator();
121        JLabel JLfont = new JLabel("Font Type");
122        toolBar.add(JLfont);
123        toolBar.addSeparator();
124        JComboBox jcb = new JComboBox(ComboStr);
125        jcb.addActionListener(new ActionListener() {
126            public void actionPerformed(ActionEvent e) {
127                theArea.append("**Combobox "+
128                    ((JComboBox)e.getSource()).getSelectedItem()+"
129                    performed!\n");
130            }
131        });
132        toolBar.add(jcb);
133
134        return toolBar;
135    }
```

```

134     }
135
136     public static void main(String[] args){
137
138         JFrame F = new JToolBar();
139         F.setSize(430,200);
140         F.addWindowListener(new WindowAdapter() {
141             public void windowClosing(WindowEvent e) {
142                 System.exit(0);
143             }
144         });
145         F.setVisible(true);
146     }
147
148     class ToolBarAction extends AbstractAction{
149
150         public ToolBarAction(String name,Icon icon){
151             super(name,icon);
152         }
153
154         public void actionPerformed(ActionEvent e){
155
156             try{
157                 theArea.append(e.getActionCommand()+"\n");
158             }catch(Exception ex){}
159         }
160     }
161 }

```

⊕ 说明:

- (1) 这个程序延伸自 JMenuItem3e.java, 将原本的程序再加上 JToolBar 类和 JToolBar 的事件处理模式而成。
- (2) 程序第 8~9 行, 预先初始化在 JComboBox 内出现的选项内容。
- (3) 程序第 21 行, 调用 buildToolBar()方法建立 JToolBar。buildToolBar()方法在程序第 85 行。
- (4) 程序第 22 行, 将构造好的 JToolBar 放置到窗口上。
- (5) 程序第 87 行, 新建一个 JToolBar。
- (6) 程序第 88 行, 利用 setFloatable()方法设置 JToolBar 能否浮动。
- (7) 程序第 90~95 行, 分别构造三个 ToolBarAction 的类组件, 准备将这三个组件放置到 JToolBar 中。ToolBarAction 类是我们自己编写的 Inner Class, 这个 Inner Class 在程序第 148~160 行。这个类继承 AbstractAction, AbstractAction 是一个抽象类, 实现 Action Interface, 而 Action Interface 又继承 ActionListener Interface, 因此可直接在 AbstractAction 中覆写 actionPerformed()方法, 处理 ActionEvent 事件。在 JToolBar 中有一个 add(Action a)方法, 只要传入 Action 参数就可以得到一个 JButton 对象, 因此在程序第 98、100 与 102 行就是以这样的方式构造出 JToolBar 上的按钮组件。

- (8) 程序第 98、100、102 行，分别将这三个 `ToolBarAction` 组件加入 `JToolBar` 中。
- (9) 程序第 99、101、103 行，利用 `setActionCommand()` 方法分别设置各个组件被触发事件后所返回的字符串信息。
- (10) 程序第 105 行，在上 `JToolBar` 加入分隔线，如同在 `JMenu` 上利用 `addSeparator()` 方法加上分隔线一样，不同的是在 `JMenu` 中分隔线是以灰色直线的样式表现，而在 `JToolBar` 中则是以一小段空白来表示。
- (11) 程序第 107~118 行，重复步骤 8 和 9 的操作加入另外三个 `ToolBarAction` 组件。
- (12) 程序第 121~122 行，在 `JToolBar` 中加入一个 `JLabel` 组件。
- (13) 程序第 124 行，利用在第 8~9 行的字符串数组建立一个 `JComboBox` 组件。
- (14) 程序第 125~130 行，将 `JComboBox` 加入事件处理模式。这里是将所选到的组件名称填入 `JTextArea` 中。
- (15) 程序第 131 行，在 `JToolBar` 中加入一个 `JComboBox` 组件。
- (16) 程序 154~159 行，对于 `JToolBar` 上按钮事件的处理是将组件的 `ActionCommand` 返回值字符串加入 `JTextArea` 中。

④ 程序运行结果如图 12-14 所示。



图 12-14

不知道大家有没有注意到在工具栏的最左边有一个很多小颗粒的区域呢？这一小块区域就是能让工具栏浮动的区域。那么该怎么让工具栏浮动呢？我们只要将鼠标指针移动到这个区域上，按下鼠标左键后拖曳到目的区域再放开左键即可。将工具栏拉动后可能会出现的情况如下：

工具栏浮动情况 1，如图 12-15 所示。



图 12-15

工具栏浮动情况 2，如图 12-16 所示。



图 12-16

工具栏之所以能够产生浮动的效果是因为我们在程序第 88 行中将工具栏的浮动与否设为“可浮动 (true)”的关系，若是我们将程序第 88 行设为不可浮动 (false) 则会如图 12-17 所示，我们会发现在工具栏左边的小颗粒区域不见了而且工具栏也不再能够被拖动。

工具栏不能浮动的情况：



图 12-17

在上面这个程序中，我们利用 `AbstractAction` 抽象类来构造 `JToolBar` 上的 `JButton` 组件，这个做法看起来较为陌生，下面我们将详细探讨其中隐含的意义。我们曾提到 `AbstractAction` 是一个抽象类，实现 `Action` 界面，而 `Action` 界面又继承 `ActionListener` 界面，因此 `AbstractAction` 类具有倾听 `ActionEvent` 的功能，我们可以覆写 `actionPerformed()` 方法来处理 `ActionEvent` 事件。然而使用 `AbstractAction` 抽象类有什么好处呢？通常一个软件在设计时会因用户习惯，产生不同的操作方式却能达到相同的功能，例如在文本编辑器中，`copy` 功能可能在菜单中出现，也可能在工具栏上出现，甚至在快捷菜单 (Popup Menu) 中出现，虽然出现的方式不一样，但都能达到 `copy` 的功能。若以我们之前的事件处理方法，我们必须为每种出现方式实现 `copy` 功能，这会造成程序代码重复性太高，不易阅读也不美观。相反的，若我们以 `AbstractAction` 抽象类来覆写 `actionPerformed()` 方法，我们只需要为每一种功能编写一次程序代码即可，因为 `JToolBar`、`JMenu`、`JPopupMenu` 类均有 `add(Action a)` 方法，可利用相同的 `AbstractAction` 类构造出个别不同的组件。在下面的范例中我们利用这个概念让用户按下“按我”选项时，均使用同一个 `actionPerformed()` 方法：

范例 `JToolBar2.java` (文件位于随书光盘目录 `exam\ch12\JToolBar2.java`)

```
1 import javax.swing.*;
2 import java.awt.*;
```

Java Swing 程序设计

```
3  import java.awt.event.*;
4
5  public class JToolBar2 extends JFrame{
6
7      JTextArea theArea = null;
8      ToolBarAction pushAction = null;
9
10     public JToolBar2(){
11
12         super("JToolBar2");
13         theArea = new JTextArea();
14         theArea.setEditable(false);
15         getContentPane().add(new JScrollPane(theArea));
16         JMenuBar MBar = new JMenuBar();
17         MBar.setOpaque(true);
18
19         pushAction = new ToolBarAction("按我", null);
20         JMenu mfile = buildFileMenu();
21         JToolBar theBar = buildToolBar();
22         this.getContentPane().add(theBar, BorderLayout.NORTH);
23
24         MBar.add(mfile);
25         setJMenuBar(MBar);
26     }
27
28     public JMenu buildFileMenu() {
29
30         JMenu thefile = new JMenu("File");
31         thefile.setMnemonic('F');
32         thefile.add(pushAction);
33
34         return thefile;
35     }
36
37     public JToolBar buildToolBar() {
38
39         JToolBar toolBar = new JToolBar();
40         toolBar.setFloatable(true);
41         toolBar.add(pushAction);
42
43         return toolBar;
44     }
45
46     public static void main(String[] args){
47
48         JFrame F = new JToolBar2();
49         F.setSize(430,200);
50         F.addWindowListener(new WindowAdapter() {
51             public void windowClosing(WindowEvent e) {
52                 System.exit(0);
53             }
54         });
55         F.setVisible(true);
56     }
57
```

```

58     class ToolBarAction extends AbstractAction{
59
60         public ToolBarAction(String name,Icon icon){
61             super(name,icon);
62         }
63
64         public void actionPerformed(ActionEvent e){
65
66             try{
67                 theArea.append("不管用哪种方式运行\"按我\", 都会出现这句!! "+"\\n");
68             }catch(Exception ex){}
69         }
70     }
71 }
72

```

说明：

- (1) 程序第32与第41行,利用ToolBarAction类分别建立JToolBar上的JButton与JMenu上的JMenuItem组件,ToolBarAction类继承了AbstractAction抽象类,并覆写actionPerformed()方法。读者在读完JPopupMenu小节后,可将JPopupMenu的功能加入此范例中。
- (2) 一般事件处理方式:先产生个别组件,再利用addActionListener()方法倾听ActionEvent事件,最后覆写actionPerformed()方法。然而以继承AbstractAction抽象类的方式处理相同的事件,可以看出程序代码精简了许多。读者自行可比较之间的差异性。

程序运行结果如图12-18所示。

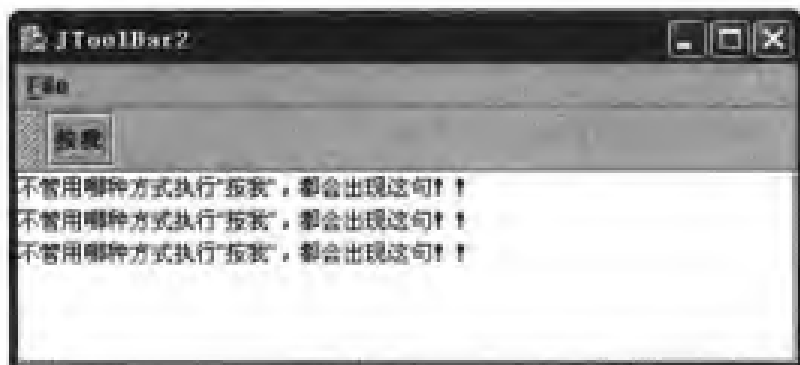


图 12-18

12-6-2 在 JToolBar 组件中加入 Tooltip

相信大多数的读者都有使用过 Word 文本编辑软件的经验,当我们在使用 Word 的工具栏时会发现有一些功能是比较不常用的,而且光看工具栏上的图标也无法立刻得知该图标代表什么功能。这时候,我们通常会把鼠标指针移到该图标上,稍微等待 1~2 秒后就会出现一个小提示让我们知道这个图标代表什么意义。这个小提示就是 Tooltip,如图 12-19 所示例中被圆圈圈起来的小黄标签部分。



图 12-19

那么我们在 `JToolBar` 中该如何使用这种功能呢？我们来看下面这个范例。这个范例延伸自 `JToolBar1.java`，因此我们只列出 `buildToolBar()` 方法的部分，其他部分的程序都没改变，请参照 `JToolBar1.java`。程序如下：

范例 `JToolBar3.java` (文件位于随书光盘目录 `exam\ch12\JToolBar3.java`)
(注意：只列出 `buildToolBar()` 方法)

```

1  public JToolBar buildToolBar() {
2
3      JToolBar toolBar = new JToolBar();
4      toolBar.setFloatable(true);
5
6      ToolBarAction tba_new = new ToolBarAction("new",
7          new ImageIcon("icons/new24.gif"));
8      ToolBarAction tba_open = new ToolBarAction("open",
9          new ImageIcon("icons/open24.gif"));
10     ToolBarAction tba_close = new ToolBarAction("close",
11         new ImageIcon("icons/close24.gif"));
12
13     JButton JB;
14     JB = toolBar.add(tba_new);
15     JB.setActionCommand("#ToolBar_NEW performed!");
16     JB.setToolTipText((String)tba_new.getValue(Action.NAME));
17     JB = toolBar.add(tba_open);
18     JB.setActionCommand("#ToolBar_OPEN performed!");
19     JB.setToolTipText((String)tba_open.getValue(Action.NAME));
20     JB = toolBar.add(tba_close);
21     JB.setActionCommand("#ToolBar_CLOSE performed!");
22     JB.setToolTipText((String)tba_close.getValue(Action.NAME));
23
24     toolBar.addSeparator();
25
26     ToolBarAction tba_B = new ToolBarAction("bold",
27         new ImageIcon("icons/bold24.gif"));
28     ToolBarAction tba_I = new ToolBarAction("italic",
29         new ImageIcon("icons/italic24.gif"));
30     ToolBarAction tba_U = new ToolBarAction("underline",
31         new ImageIcon("icons/underline24.gif"));
32     JB = toolBar.add(tba_B);
33     JB.setActionCommand("#ToolBar_Bold performed!");
34     JB.setToolTipText((String)tba_B.getValue(Action.NAME));
35     JB = toolBar.add(tba_I);
36     JB.setActionCommand("#ToolBar_Italic performed!");
37     JB.setToolTipText((String)tba_I.getValue(Action.NAME));
38     JB = toolBar.add(tba_U);
39     JB.setActionCommand("#ToolBar_Underline performed!");

```

```

40     JB.setToolTipText((String)tba_U.getValue(Action.NAME));
41
42     toolBar.addSeparator();
43     JLabel JLfont = new JLabel("Font Type");
44     toolBar.add(JLfont);
45     toolBar.addSeparator();
46     JComboBox jcb = new JComboBox(ComboStr);
47     jcb.addActionListener(new ActionListener() {
48         public void actionPerformed(ActionEvent e) {
49             theArea.append("*Combobox "+
50 ((JComboBox)e.getSource()).getSelectedItem()+
51 " performed!\n");
52         });
53     toolBar.add(jcb);
54
55     return toolBar;
56 }

```

说明：

- (1) 在 JToolBar 中加入 Tooltip 只需要利用 setToolTipText()这个方法即可。
- (2) 程序第 16、19、22、34、37、40 行，利用 setToolTipText()方法分别设置在工具栏上各个图标提示文字。在这里我们还利用 getValue()方法取得各个 ToolBarAction 类的名称来当作 Tooltip 显示的字符串。

程序运行结果如图 12-20 所示。



图 12-20

12-7 使用 JPopupMenu 组件

JPopupMenu 的类层次结构图：

```

java.lang.Object
--java.awt.Component
--java.awt.Container
--javax.swing.JComponent
--javax.swing.JPopupMenu

```

JPopupMenu 是一种特别形式的 Menu，其性质与 Menu 几乎完全相同，但是 PopupMenu 并不固定在窗口中的任何一个位置，而是由鼠标指针和系统判断决定 PopupMenu 要出现在哪里。相信大家在使用许多软件时都用到过 PopupMenu 的功能，例如在使用 Word 文本编辑软

件时，当我们在编辑区域的任意处按下鼠标右键，就会跳出一个 PopupMenu，如图 12-21 所示。



图 12-21

这就是一个标准的 PopupMenu 范例，那么我们在 Swing 中该怎样来建立一个 PopupMenu 呢？先来看看 JPopupMenu 的构造函数，如表 12-7 所示。

表 12-7

JPopupMenu 的构造函数	
JPopupMenu()	建立一个新的 JPopupMenu
JPopupMenu(String label)	建立一个指定标题的 JPopupMenu

12-7-1 构造 JPopupMenu 组件

由上表中我们可以知道 JPopupMenu 的构造方法并不复杂，两种构造方法的差异只在于是否赋予 JPopupMenu 一个标题文字。我们马上来看一个范例，了解该如何应用 JPopupMenu：

范例 JPopupMenu1.java (文件位于随书光盘目录 exam\ch12\JPopupMenu1.java)

```

1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4  import javax.swing.event.*;
5  import javax.swing.border.*;
6
7  public class JPopupMenu1 extends JFrame{
8
9      JTextArea theArea = null;
10     static final String ComboStr[] = {"Times New Roman","Dialog","宋体",
11     "黑体","楷体"};
12     JPopupMenu Popup = null;
13
14     public JPopupMenu1(){
15
16         super("JPopupMenu1");

```

```

17     theArea = new JTextArea();
18     theArea.setEditable(false);
19     this.getContentPane().add(new JScrollPane(theArea), BorderLayout.CENTER);
20     JMenuBar MBar = new JMenuBar();
21     MBar.setOpaque(true);
22     JMenu mfile = buildFileMenu();
23     JToolBar theBar = buildToolBar();
24     this.getContentPane().add(theBar, BorderLayout.NORTH);
25     PopupPanel pp = new PopupPanel();
26     this.getContentPane().add(pp, BorderLayout.SOUTH);
27
28     MBar.add(mfile);
29     setJMenuBar(MBar);
30 }
31
32 public JMenu buildFileMenu() {
33
34     JMenu thefile = new JMenu("File");
35     thefile.setMnemonic('F');
36
37     JMenuItem newf = new JMenuItem("New",
38         new ImageIcon("icons/new24.gif"));
39     JMenuItem open = new JMenuItem("Open",
40         new ImageIcon("icons/open24.gif"));
41     JMenuItem close = new JMenuItem("Close",
42         new ImageIcon("icons/close24.gif"));
43     JMenuItem quit = new JMenuItem("Exit",
44         new ImageIcon("icons/exit24.gif"));
45
46     newf.setMnemonic('N');
47     open.setMnemonic('O');
48     close.setMnemonic('L');
49     quit.setMnemonic('X');
50
51     newf.setAccelerator( KeyStroke.getKeyStroke
52         ('N', java.awt.Event.CTRL_MASK, false) );
53     open.setAccelerator( KeyStroke.getKeyStroke
54         ('O', java.awt.Event.CTRL_MASK, false) );
55     close.setAccelerator( KeyStroke.getKeyStroke
56         ('L', java.awt.Event.CTRL_MASK, false) );
57     quit.setAccelerator( KeyStroke.getKeyStroke
58         ('X', java.awt.Event.CTRL_MASK, false) );
59
60     newf.addActionListener(new ActionListener() {
61         public void actionPerformed(ActionEvent e) {
62             theArea.append("- MenuItem New Performed -\n");
63         }
64     });
65
66     open.addActionListener(new ActionListener() {
67         public void actionPerformed(ActionEvent e) {
68             theArea.append("- MenuItem Open Performed -\n");
69         }
70     });

```



Java Swing 程序设计

```
70
71     close.addActionListener(new ActionListener() {
72         public void actionPerformed(ActionEvent e) {
73             theArea.append("~ MenuItem Close Performed ~\n");
74         }
75     });
76
77     quit.addActionListener(new ActionListener() {
78         public void actionPerformed(ActionEvent e) {
79             System.exit(0);
80         }
81     });
82
83     thefile.add(newf);
84     thefile.add(open);
85     thefile.add(close);
86     thefile.addSeparator();
87     thefile.add(quit);
88
89     return thefile;
90 }
91
92 public JToolBar buildToolBar() {
93     JToolBar toolBar = new JToolBar();
94     toolBar.setFloatable(true);
95
96     ToolBarAction tba_new = new ToolBarAction("new",
97         new ImageIcon("icons/new24.gif"));
98     ToolBarAction tba_open = new ToolBarAction("open",
99         new ImageIcon("icons/open24.gif"));
100    ToolBarAction tba_close = new ToolBarAction("close",
101        new ImageIcon("icons/close24.gif"));
102
103    JButton JB;
104    JB = toolBar.add(tba_new);
105    JB.setActionCommand("#ToolBar_NEW performed!");
106    JB.setToolTipText((String)tba_new.getValue(Action.NAME));
107    JB = toolBar.add(tba_open);
108    JB.setActionCommand("#ToolBar_OPEN performed!");
109    JB.setToolTipText((String)tba_open.getValue(Action.NAME));
110    JB = toolBar.add(tba_close);
111    JB.setActionCommand("#ToolBar_CLOSE performed!");
112    JB.setToolTipText((String)tba_close.getValue(Action.NAME));
113
114    toolBar.addSeparator();
115
116    ToolBarAction tba_B = new ToolBarAction("bold",
117        new ImageIcon("icons/bold24.gif"));
118    ToolBarAction tba_I = new ToolBarAction("italic",
119        new ImageIcon("icons/italic24.gif"));
120    ToolBarAction tba_U = new ToolBarAction("underline",
121        new ImageIcon("icons/underline24.gif"));
122    JB = toolBar.add(tba_B);
123    JB.setActionCommand("#ToolBar_Bold performed!");
```

```

123     JB.setToolTipText((String)tba_B.getValue(Action.NAME));
124     JB = toolBar.add(tba_I);
125     JB.setActionCommand("#ToolBar_Italic performed!");
126     JB.setToolTipText((String)tba_I.getValue(Action.NAME));
127     JB = toolBar.add(tba_U);
128     JB.setActionCommand("#ToolBar_Underline performed!");
129     JB.setToolTipText((String)tba_U.getValue(Action.NAME));
130
131     toolBar.addSeparator();
132     JLabel JLfont = new JLabel("Font Type");
133     toolBar.add(JLfont);
134     toolBar.addSeparator();
135     JComboBox jcb = new JComboBox(ComboStr);
136     jcb.addActionListener(new ActionListener() {
137         public void actionPerformed(ActionEvent e) {
138             theArea.append("*Combobox "+
139 ((JComboBox)e.getSource()).getSelectedItem()+
140 " performed!\n");
141         });
142     toolBar.add(jcb);
143
144     return toolBar;
145 }
146
147 public static void main(String[] args){
148
149     JFrame F = new JPopupMenu1();
150     F.setSize(430,200);
151     F.addWindowListener(new WindowAdapter() {
152         public void windowClosing(WindowEvent e) {
153             System.exit(0);
154         }
155     });
156     F.setVisible(true);
157 }
158
159 class ToolBarAction extends AbstractAction{
160
161     public ToolBarAction(String name,Icon icon){
162         super(name,icon);
163     }
164
165     public void actionPerformed(ActionEvent e){
166
167         try{
168             theArea.append(e.getActionCommand()+"\n");
169         }catch(Exception ex){}
170     }
171 }
172
173 class PopupPanel extends JPanel implements
174 MouseListener,PopupMenuListener,ActionListener{
175

```

```
176     public PopupPanel(){
177
178         Popup = new JPopupMenu();
179
180         JMenuItem theItem;
181         Popup.add(theItem = new JMenuItem("Cut",
182         new ImageIcon("icons/cut24.gif")));
183         theItem.addActionListener(this);
184         Popup.add(theItem = new JMenuItem("Copy",
185         new ImageIcon("icons/copy24.gif")));
186         theItem.addActionListener(this);
187         Popup.add(theItem = new JMenuItem("Paste",
188         new ImageIcon("icons/paste24.gif")));
189         theItem.addActionListener(this);
190         Popup.addSeparator();
191         Popup.add(theItem = new JMenuItem("Page Setup..."));
192         theItem.addActionListener(this);
193
194         Popup.setBorder(new BevelBorder(BevelBorder.RAISED));
195
196         Popup.addPopupMenuListener(this);
197         addMouseListener(this);
198     }
199
200
201     public void mouseClicked(MouseEvent me){ checkPopup(me);}
202     public void mousePressed(MouseEvent me){ checkPopup(me);}
203     public void mouseReleased(MouseEvent me){ checkPopup(me);}
204     public void mouseEntered(MouseEvent me){}
205     public void mouseExited(MouseEvent me){}
206
207     private void checkPopup(MouseEvent me){
208         if(me.isPopupTrigger()){
209             Popup.show(me.getComponent(), me.getX(), me.getY());
210         }
211     }
212
213     public void popupMenuWillBecomeVisible(PopupMenuEvent pme){
214         theArea.append("-PopupMenu Visibel!\n");
215     }
216     public void popupMenuWillBecomeInvisible(PopupMenuEvent pme){
217         theArea.append("-PopupMenu Invisibel!\n");
218     }
219     public void popupMenuCanceled(PopupMenuEvent pme){
220         theArea.append("-PopupMenu Hidden!\n");
221     }
222
223     public void actionPerformed(ActionEvent ae){
224         theArea.append("+Popup "+ae.getActionCommand()+
225         " performed!\n");
226     }
227 }
228 }
229
```

说明:

- (1) 由于 JPopupMenu 只是一种 Menu 的组件, 因此在使用 JPopupMenu 时我们都需要一个 Container 来放置 JPopupMenu, 在这里我们选择使用 JPanel 组件来作为 JPopupMenu 的 Container。
- (2) 程序第 12 行, 声明一个全局 (Global) 并设置初始值为 null 的 JPopupMenu 组件。
- (3) 程序第 25 行, 新建一个 PopupPanel 组件来放置 JPopupMenu。
- (4) 程序第 26 行, 将 PopupPanel 组件放置到窗口中。
- (5) 程序第 32~171 行, 分别建立程序中菜单、工具栏和事件处理模式的部分, 我们在此不再加以说明。
- (6) 程序第 173~227 行, 建立 Inner Class PopupPanel。这个类继承了 JPanel 并实作 MouseListener、PopupMenuListener 和 ActionListener 这 3 个 Listener。MouseListener 为处理鼠标事件的 Listener; PopupMenuListener 为 Popup 菜单事件的 Listener; ActionListener 则是大家最熟悉用来处理 JMenuItem 事件的 Listener。
- (7) 程序第 178 行, 新建一个 JPopupMenu。
- (8) 程序第 180~192 行, 分别建立四个 JMenuItem 和其事件处理模式并将 JMenuItem 加入 JPopupMenu 中。其中程序第 181、184、187、191 行, 在建立 JMenuItem 组件, 程序第 183、186、189、192 行, 分别将各个 JMenuItem 组件加入事件处理模式, 程序第 223~226 行, 当某个 JMenuItem 被触发后会在 JTextArea 中加入一个信息。程序第 190 行, 在 JPopupMenu 中加入分隔线。我们在前面提到过 JPopupMenu 和 Menu 大致上的性质是一样的, 所以在 JPopupMenu 中一样是以 JMenuItem 来当作选项的组件。
- (9) 程序第 194 行, 利用 setBorder() 方法将 BevelBorder 类当作参数传入, 使 JPopupMenu 产生立体边框浮起的效果。我们也可以将参数设置改为 BevelBorder.LOWERED 使得 JPopupMenu 产生立体边框凹陷的效果。
- (10) 程序第 196 行, 将 JPopupMenu 组件加入 PopupMenu 事件处理模式中。
- (11) 程序第 197 行, 将 PopupPanel 类加入 Mouse 事件处理模式中。
- (12) 程序第 201~205 行, 由于 PopupPanel 这个 Inner Class 中实作了 MouseListener, 因此需要覆写 mouseClicked()、mousePressed()、mouseReleased()、mouseEntered() 和 mouseExited() 这 5 个方法。在这个范例中当 mouseClicked()、mousePressed() 和 mouseReleased() 这 3 个方法被触发时会调用 checkPopup() 方法来判断是否要将 PopupMenu 输出来。
- (13) 程序第 207~211 行, 为 checkPopup() 方法。其中, 在程序第 208 行利用 isPopupTrigger() 方法判断 PopupMenu 是否已经输出。若没有, 则在程序第 209 行利用 show() 方法将 PopupMenu 显示到目前鼠标指针 (利用 getX() 和 getY() 方法得知) 的位置上。
- (14) 程序第 213~221 行, 由于 PopupPanel 这个 Inner Class 中实作了 PopupMenuListener, 因此需要覆写 popupMenuWillBecomeVisible()、popupMenuWillBecomeInvisible() 和 popupMenuCanceled() 这 3 个方法。在这个范例中当这 3 个方法被触发后会在 JTextArea 中加入一条信息。

◆ 程序运行结果如图 12-22 所示。



图 12-22

在这个程序的运行结果中读者可能会发现 JPopupMenu 只有在 JTextArea 的下缘,也就是 JPanel 的部分才有作用,因为我们并没有在 JTextArea 中加入 addMouseListener()方法。您可以在 PopupPanel()构造函数中加入“theArea.addMouseListener(this);”这行,就能够解决在 JTextArea 上无法显示 JPopupMenu 的问题。

12-8 本章总结

在此章中,我们先介绍了构成一个菜单不可缺少的三种组件——JMenuBar、JMenu 和 JMenuItem,并且在 JMenuItem 中介绍了 JMenu 与 JMenuItem 的快捷键和快捷键的设置方式,让菜单的功能和外观设计更为完整。然后我们介绍了两种特殊的 JMenuItem——JCheckBoxMenuItem 与 JRadioButtonMenuItem,这两者除了外观不同外,其余性质几乎完全相同。接下来我们看到如何构造一个工具栏并且了解如何使用工具栏上的事件处理机制,然后再加上 JToolTip 的提示功能使工具栏的构造更完备。最后我们介绍 JPopupMenu 组件来增加软件的易用性。在读完本章后,相信读者一定能设计出功能完备的菜单列表。

12-9 本章习题

1. 利用 JMenu 的特性构造一个 3 层次式的 JMenu。
2. JCheckBoxMenuItem 是否也可以变成单选组件?若可以,应该怎么做?
3. 试实现相关范例来说明使用 AbstractAction 抽象类的时机与意义。
4. 试在同一个 JTextArea 内产生两个不同的 JPopupMenu 组件。

13

文件选择对话框、颜色选择对话框、分隔线的使用与介绍

在本章中我们将介绍文件选择对话框(JFileChooser)的各种功能,包括基本文件对话框的建置、文件类型的过滤、文件显示图式的改变等等。友善的文件选择对话框可以让用户更轻松地进行文件的存取操作。之后我们将介绍颜色选择对话框(JColorChooser)的功能,让用户可以轻松地为自己的组件选择所喜欢的颜色,并让用户知道如何将颜色选择面板置于一般容器中,或是对颜色选择面板作新增、删除的操作。最后,我们简单地介绍了分隔线(JSeparator)的使用,让用户更能了解分隔线的功能与作用。

深入淺出

13-1 使用 JFileChooser 组件

JFileChooser 的类层次结构图:

```
java.lang.Object
--java.awt.Component
--java.awt.Container
--javax.swing.JComponent
-- javax.swing.JFileChooser
```

当我们在处理窗口上的一些操作，特别是处理文本部分时，例如在一个文本编辑器上输入了一段文字，我们可能希望将此段文字存储起来，供以后方便调用，此时系统应当提供一个存储文件的对话框，将此段文字存到一个自定义或默认文件名的文件中。同样，当我们要打开某个文件时，系统也应当提供开文件的功能，让我们选择所要打开的文件。在 Java 中这些操作都可以由 JFileChooser 组件来达成。这个组件提供了打开文件存盘的窗口功能，也提供了显示特定类型文件图标的功能，亦能针对某些文件类型做过滤的操作。如果您的系统需要对某些文件或文件夹做操作，JFileChooser 组件可以让您轻松地做出漂亮的用户界面。在这里读者要注意的是，JFileChooser 本身不提供文件读取或存盘的功能，这些功能必须由您自行实现。事实上，JFileChooser 本身只是一个对话框类型，它也是依附在 JDialog 的结构上，因此它只是一个针对文件操作的对话框，当然本身也就不会有文件读取或存盘的功能。下面我们来看 JFileChooser 的构造函数，如表 13-1 所示。

表 13-1

JFileChooser 的构造函数	
JFileChooser()	建立一个 JFileChooser 对象，默认的文件对话框路径是用户的家目录 (Home Directory)，例如在 Windows 2000 中的 Administrator 的家目录是在 C:\Documents and Settings\Administrator 中
JFileChooser(File currentDirectory)	建立一个 JFileChooser 对象，并以 File 所在位置为文件对话框的打开路径
JFileChooser(File currentDirectory, FileSystemView fsv)	建立一个 JFileChooser 对象，以 File 所在位置为文件对话框的打开路径并设置文件图标查看方式
JFileChooser(FileSystemView fsv)	建立一个 JFileChooser 对象，并设置文件图标查看方式
JFileChooser(String currentDirectoryPath)	建立一个 JFileChooser 对象，并设置文件对话框的打开路径
JFileChooser(String currentDirectoryPath, FileSystemView fsv)	建立一个 JFileChooser 对象，并设置文件对话框的打开路径与文件图标查看方式

13-1-1 建立一个简单的 JFileChooser 对话框

介绍完 JFileChooser 构造函数后，我们来实现一个简单的范例。这个范例可以让用户在 JTextArea 上输入文字，输入完后按下“存储文件”按钮就可以打开 JFileChooser 存储文件对话框，用户可以输入要存储的文件文件名，按下“保存”按钮就可以存储文件。若用户要打开某个文件内容，只需要按下“打开文件”按钮，就会出现 JFileChooser 打开文件对话框，

用户选择好所要打开的文件就可以将数据读入 JTextArea 中。

在这个范例中,我们使用 JFileChooser 的 showOpenDialog()或 showSaveDialog()方法来打开文件对话框,这两种方法在用户按下按钮或关闭对话框时会返回一个整数值,这个整数值类型有 3 种,分别是:

JFileChooser.CANCEL_OPTION: 表示用户按下取消按钮。

JFileChooser.APPROVE_OPTION: 表示用户按下确定按钮。

JFileChooser.ERROR_OPTION: 表示有错误产生或是对话框非正常关闭。

利用这 3 个整数值我们就能判断出用户到底在对话框中做了什么操作,并加以处理。例如当用户选择了文件并按下确定键后,我们就可以利用 getSelectedFile()方法取得文件对象,利用这个文件对象我们就能够取得文件名称 (getName()) 与文件路径 (getPath())。

```

1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4  import java.io.*;
5
6  class FileChooserDemo1 implements ActionListener
7  {
8      JFrame f = null;
9      JLabel label = null;
10     JTextArea textarea = null;
11     JFileChooser fileChooser = null;
12
13     public FileChooserDemo1()
14     {
15         f = new JFrame("FileChooser Example");
16         Container contentPane = f.getContentPane();
17         textarea = new JTextArea();
18         JScrollPane scrollPane = new JScrollPane(textarea);
19         scrollPane.setPreferredSize(new Dimension(350,300));
20
21         JPanel panel = new JPanel();
22         JButton b1 = new JButton("新建文件");
23         b1.addActionListener(this);
24         JButton b2 = new JButton("存储文件");
25         b2.addActionListener(this);
26         panel.add(b1);
27         panel.add(b2);
28
29         label = new JLabel(" ",JLabel.CENTER);
30
31         fileChooser = new JFileChooser("D:\\");
32
33         contentPane.add(label,BorderLayout.NORTH);
34         contentPane.add(scrollPane,BorderLayout.CENTER);
35         contentPane.add(panel,BorderLayout.SOUTH);
36
37         f.pack();
38         f.setVisible(true);
39
40         f.addWindowListener(new WindowAdapter() {

```

```
41         public void windowClosing(WindowEvent e) {
42             System.exit(0);
43         }
44     });
45 }
46
47 public static void main(String[] args) {
48     new FileChooserDemol();
49 }
50
51 public void actionPerformed(ActionEvent e)
52 {
53     File file = null;
54     int result;
55
56     if (e.getActionCommand().equals("新建文件"))
57     {
58         fileChooser.setApproveButtonText("确定");
59         fileChooser.setDialogTitle("打开文件");
60         result = fileChooser.showOpenDialog(f);
61
62         textarea.setText("");
63
64         if (result == JFileChooser.APPROVE_OPTION)
65         {
66             file = fileChooser.getSelectedFile();
67             label.setText("您选择打开的文件名称为: "+file.getName());
68         }
69         else if (result == JFileChooser.CANCEL_OPTION)
70         {
71             label.setText("您没有选择任何文件");
72         }
73
74         FileInputStream fileInStream = null;
75
76         if (file != null)
77         {
78             try{
79                 fileInStream = new FileInputStream(file);
80             }catch (FileNotFoundException fe){
81                 label.setText("File Not Found");
82                 return;
83             }
84
85             int readbyte;
86
87             try{
88                 while( (readbyte = fileInStream.read()) != -1)
89                 {
90                     textarea.append(String.valueOf((char) readbyte));
91                 }
92             }catch (IOException ioe){
93                 label.setText("读取文件错误");
94             }
95             finally{
```

```

96         try{
97             if(fileInStream != null)
98                 fileInStream.close();
99         }catch(IOException ioe2){}
100     }
101 }
102 }
103
104 if (e.getActionCommand().equals("存储文件"))
105 {
106     result = fileChooser.showSaveDialog(f);
107     file = null;
108     String fileName;
109
110     if (result == JFileChooser.APPROVE_OPTION)
111     {
112         file = fileChooser.getSelectedFile();
113         label.setText("您选择存储的文件名称为: "+file.getName());
114     }
115     else if(result == JFileChooser.CANCEL_OPTION)
116     {
117         label.setText("您没有选择任何文件");
118     }
119
120     FileOutputStream fileOutputStream = null;
121
122     if(file != null)
123     {
124         try{
125             fileOutputStream = new FileOutputStream(file);
126         }catch(FileNotFoundException fe){
127             label.setText("File Not Found");
128             return;
129         }
130
131         String content = textarea.getText();
132
133         try{
134             fileOutputStream.write(content.getBytes());
135         }catch(IOException ioe){
136             label.setText("写入文件错误");
137         }
138         finally{
139             try{
140                 if(fileOutputStream != null)
141                     fileOutputStream.close();
142             }catch(IOException ioe2){}
143         }
144     }
145 }
146 }
147 }

```

⊕ 说明:

- (1) 程序第 31 行, 建立一个 FileChooser 对象, 并指定 D 盘的根目录为默认文件对话框路径。
- (2) 程序第 58~60 行, 当用户按下“打开文件”按钮时, JFileChooser 的 showOpenDialog() 方法会输出打开文件对话框, 并利用 setApproveButtonText() 方法取代按钮上“Open”文字; 以 setDialogTitle() 方法设置打开文件对话框 Title 的名称。当选择完后, 会将选择结果存到 result 变量中。
- (3) 程序第 64~72 行, 当用户按下打开文件对话框的“打开”按钮后, 我们就可以利用 getSelectedFile() 方法取得文件对象。若是用户按下开文件对话框的“撤消”按钮, 则将在 label 上显示“您没有选择任何文件”字样。
- (4) 程序第 79 行, 利用 FileInputStream 将文件内容放入此数据流中以便读取。
- (5) 程序第 88~91 行, 以 read() 方法读取 FileInputStream 对象的内容, 当返回值为 -1 时表示读完此数据流。将所读到的字符显示在 textarea 中。
- (6) 程序第 95~100 行, 释放 FileInputStream 对象, 避免资源的浪费。
- (7) 程序第 104~146 行, 实作写入文件的功能。
- (8) 程序第 110~118 行, 当用户没有选择文件, 而是自己键入文件名称时, 系统会自动以此文件名建立新文件。
- (9) 程序第 120 行, 写入文件我们使用 FileOutputStream。在这个范例中, 我们写入文件的方式是将之前的内容清除再重新写入。若您想把新增的内容加在原有文件内容的后面, 您可以使用 FileOutputStream(String name, Boolean append) 这个构造函数。

⊕ 程序运行结果如下:

当用户在 textarea 上输入完文字并按下“存储文件”按钮时所显示的窗口, 如图 13-1 所示。



图 13-1

用户输入要存的文件名为 test.txt，若系统没有此文件则会自动建立。

存储完后我们关闭程序再重新打开，这时候我们按下“打开文件”按钮，如图 13-2 所示。



图 13-2

此时您可以看到 test.txt 文件已经出现在我们的系统上，选择此文件后按下“打开”按钮就可以看到这个文件的内容了，如图 13-3 所示。



图 13-3

13-1-2 建立可选择文件类型的 JFileChooser 对话框

当您专为某种文件类型设计一套软件时，为了用户打开文件存盘方便，我们通常会在文件对话框中过滤掉无关的文件类型，让用户能够快速的选择出想要的文件类型数据。例如在 Word 软件中，当我们按下“另存新文件”选项时，所出现的文件对话框将会以“.doc”扩展名当作默认的文件存储类型，如图 13-4 所示。



图 13-4

如果您所设计的软件可以支持多种类型的文件操作，您应该设计让用户可以选择使用哪一种类型的文件，如图 13-5 所示。



图 13-5

若您想在 Java 的文件对话框中做到这样的功能，您必须实现 `FileFilter` 这个抽象类。此抽象类里面定义了两种方法，分别是 `accept(File f)` 与 `getDescription()`。当目录里的文件与设置的文件类型相符时，`accept()` 方法就会返回 `true`，并将此文件显示在文件对话框中。而 `getDescription()` 方法则是对此文件类型的描述，可由程序设计者自定义，如“Java Source File (*.java)”等等。要设置选择文件类型对话框您可以使用 `JFileChooser` 的 `addChoosableFileFilter()` 方法或是 `setFileFilter()` 方法。下面这个例子中我们实现 `FileFilter` 的功能，让用户打开文件时可以选择显示所有文件，或是 *.java 文件，亦或是 *.class 文件。

```

1  import java.awt.*;
2  import javax.swing.*;
3  import java.awt.event.*;
4  import java.io.*;
5  import javax.swing.filechooser.*;
6
7  public class FileFilterDemo implements ActionListener
8  {
9      JFrame f = null;
10     JLabel label = null;
11     JFileChooser fileChooser = null;
12
13     public FileFilterDemo()
14     {
15         f = new JFrame("FileFilter Demo");
16         Container contentPane = f.getContentPane();
17
18         JButton b = new JButton("打开文件");

```

```

19         b.addActionListener(this);
20
21         label = new JLabel(" ",JLabel.CENTER);
22         label.setPreferredSize(new Dimension(150,30));
23         contentPane.add(label,BorderLayout.CENTER);
24         contentPane.add(b,BorderLayout.SOUTH);
25
26         f.pack();
27         f.setVisible(true);
28
29         f.addWindowListener(new WindowAdapter() {
30             public void windowClosing(WindowEvent e) {
31                 System.exit(0);
32             }
33         });
34     }
35
36     public static void main(String[] args)
37     {
38         new FileFilterDemo();
39     }
40
41     public void actionPerformed(ActionEvent e)
42     {
43         fileChooser = new JFileChooser("c:\\\\winnt");
44         fileChooser.addChoosableFileFilter(new JAVAFileFilter("class"));
45         fileChooser.addChoosableFileFilter(new JAVAFileFilter("java"));
46         int result = fileChooser.showOpenDialog(f);
47         if(result == JFileChooser.APPROVE_OPTION)
48         {
49             File file = fileChooser.getSelectedFile();
50             label.setText("您选择了: "+file.getName()+"文件");
51         }else if (result == fileChooser.CANCEL_OPTION){
52             label.setText("您没有选取文件");
53         }
54     }
55 }
56
57 class JAVAFileFilter extends FileFilter
58 {
59     String ext;
60
61     public JAVAFileFilter(String ext)
62     {
63         this.ext = ext;
64     }
65
66     public boolean accept(File file)
67     {
68         if (file.isDirectory())
69             return true;
70
71         String fileName = file.getName();
72         int index = fileName.lastIndexOf('.');
73

```



```
74         if (index > 0 && index < fileName.length()-1) {
75             //表示文件名称不为“.xxx”与“xxx.”之类型
76             String extension = fileName.substring(index+1).toLowerCase();
77             if (extension.equals(ext))
78                 return true;
79         }
80         return false;
81     }
82
83     public String getDescription(){
84         if (ext.equals("java"))
85             return "JAVA Source File (*.java)";
86         if (ext.equals("class"))
87             return "JAVA Class File (*.class)";
88         return "";
89     }
90 }
```

⊕ 说明:

- (1) 程序第4~5行,由于我们在程序中要使用到File与FileFilter对象,因此要importFile与FileFilter这两个类。
- (2) 程序第41~54行,处理用户按下“打开文件”按钮的事件。
- (3) 程序第43行,建立一个JFileChooser对象,并设置“c:\\winnt”为打开文件对话框的默认路径。
- (4) 程序第44~45行,利用addChoosableFileFilter()方法加入要过滤的文件类型。使用addChoosableFileFilter()可以加入多种文件类型,若您只需要过滤出一种文件类型,可以使用setFileFilter()方法。
- (5) 程序第57~90行,以JavaFileFilter类继承FileFilter抽象类,并实现accept()与getDescription()方法。
- (6) 程序第66~81行,在accept()方法中,当程序所选取的是一个目录而不是文件时,我们返回true值,表示将此目录显示出来。程序第77行,若所选取的文件扩展名等于我们所设置要显示的扩展名(即变量ext值),则返回true,表示将此文件显示出来,否则返回false。
- (7) 程序第83~89行,实现getDescription()方法,返回描述文件的说明字符串。

⊕ 程序运行结果如图13-6所示。



图 13-6

按下“打开文件”按钮后,如图13-7所示。



图 13-7

从上图中我们可以看出, 由于我们选择显示 java 文件类型, 因此文件对话框中出现的都是 java 文件, 而且此对话框可以让我们选择 3 种文件显示方式, 分别是显示所有文件(*.*)、*.java 文件、与 *.class 文件。

您可以使用 Java 的 Look and Feel 功能, 使得对话框能够长得很像原有操作系统默认的类型, 例如我们如果要使对话框类似 Windows 的窗口类型, 可以在程序第 43 行前加入下列几行:

```
try {
    UIManager.setLookAndFeel(
        "com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
} catch (Exception e) {
    System.out.println ("Look and Feel Exception");
    System.exit(0);
}
```

运行结果将变成如图 13-8 所示。

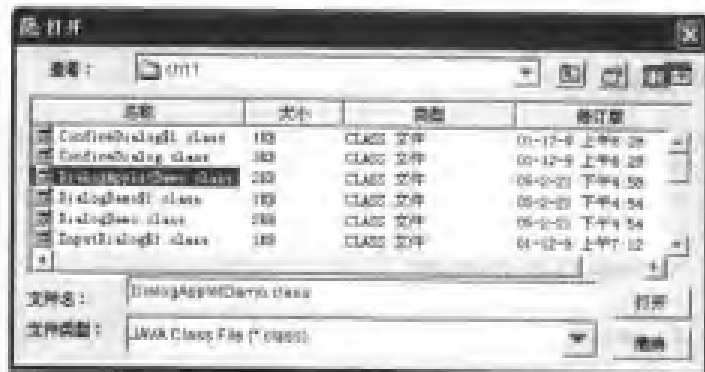


图 13-8

有没有很熟悉的感觉呢! Java 的 Look and Feel 功能我们将在第 15 章中介绍。

13-1-3 建立具有特殊文件类型图标 of JFileChooser

在上个范例中, 读者可以发现若您选择显示所有文件时, 文件类型图标不会因扩展名的

不同而有所区别，这样可能造成用户混淆或是使用上的不方便，如图 13-9 所示。



图 13-9

要解决这个问题，您必须再实现 `FileView` 这个抽象类，该抽象类定义了 5 种方法，如表 13-2 所示。

表 13-2

FileView 方法	
String	<code>getDescription(File f)</code> 返回对这个文件的描述，如这是一张风景图片等等
Icon	<code>getIcon(File f)</code> 返回文件 Icon 图标
String	<code>getName(File f)</code> 返回文件名
String	<code>getTypeDescription(File f)</code> 返回文件类型描述，如：“Java Source File”等等
Boolean	<code>isTraversable(File f)</code> 返回目录是否可浏览

当您实现好这 5 种方法后，就可以利用 `JFileChooser` 的 `setFileView()` 方法来设置文件类型图标。我们来看下面的范例：

```

1  import java.awt.*;
2  import javax.swing.*;
3  import java.awt.event.*;
4  import java.io.File;
5  import javax.swing.filechooser.*;
6
7  public class FileIconFilterDemo implements ActionListener
8  {
9      JFrame f = null;
10     JLabel label = null;
11     JFileChooser fileChooser = null;
12
13     public FileIconFilterDemo()
14     {
15         f = new JFrame("FileIconFilter Demo");
16         Container contentPane = f.getContentPane();

```

```

17
18     JButton b = new JButton("打开文件");
19     b.addActionListener(this);
20
21     label = new JLabel(" ", JLabel.CENTER);
22     label.setPreferredSize(new Dimension(150, 30));
23     contentPane.add(label, BorderLayout.CENTER);
24     contentPane.add(b, BorderLayout.SOUTH);
25
26     f.pack();
27     f.setVisible(true);
28
29     f.addWindowListener(new WindowAdapter() {
30         public void windowClosing(WindowEvent e) {
31             System.exit(0);
32         }
33     });
34 }
35
36 public static void main(String[] args)
37 {
38     new FileIconFilterDemo();
39 }
40
41 public void actionPerformed(ActionEvent e)
42 {
43     fileChooser = new JFileChooser("c:\\winnt");
44     fileChooser.addChoosableFileFilter(new JAVAFileFilter("class"));
45     fileChooser.addChoosableFileFilter(new JAVAFileFilter("java"));
46     fileChooser.setFileView(new FileIcon());
47     int result = fileChooser.showOpenDialog(f);
48     if(result == JFileChooser.APPROVE_OPTION)
49     {
50         File file = fileChooser.getSelectedFile();
51         label.setText("您选择了: "+file.getName()+"文件");
52     }else if (result == fileChooser.CANCEL_OPTION){
53         label.setText("您没有选取文件");
54     }
55 }
56 }
57
58 class JAVAFileFilter extends FileFilter
59 {
60     String ext;
61
62     public JAVAFileFilter(String ext)
63     {
64         this.ext = ext;
65     }
66
67     public boolean accept(File file)
68     {
69         if (file.isDirectory())
70             return true;
71

```

```
72     String fileName = file.getName();
73     int index = fileName.lastIndexOf('.');
74
75     if (index > 0 && index < fileName.length()-1) {
76         //表示文件名称不为“.xxx”与“xxx.”之类型
77         String extension = fileName.substring(index+1).toLowerCase();
78         if (extension.equals(ext))
79             return true;
80     }
81     return false;
82 }
83
84 public String getDescription(){
85     if (ext.equals("java"))
86         return "JAVA Source File (*.java)";
87     if (ext.equals("class"))
88         return "JAVA Class File (*.class)";
89     return "";
90 }
91 }
92
93 class FileIcon extends FileView
94 {
95     public String getName(File f) {
96         return null;    //JAVA Look and Feel 功能会处理掉这个项目
97                         //一般而言可以使用 f.getName() 当返回值
98     }
99
100    public String getDescription(File f) {
101        return null;    //JAVA Look and Feel 功能会处理掉这个项目
102                        //您也可以自己设置对此图片的描素，如：
103                        //这是一张风景图片等等
104    }
105
106    public String getTypeDescription(File f)
107    {
108        String extension = getExtensionName(f);
109        if(extension.equals("java"))
110            return "JAVA Source File";
111        if(extension.equals("class"))
112            return "JAVA Class File";
113        return "";
114    }
115
116    public Icon getIcon(File f)
117    {
118        String extension = getExtensionName(f);
119        if(extension.equals("java"))
120            return new ImageIcon("java.gif");
121        if(extension.equals("class"))
122            return new ImageIcon("class.gif");
123        return null;
124    }
125
126    public Boolean isTraversable(File f) {
```

```

127     return null;    //JAVA Look and Feel 功能会处理掉这个项目
128                     //若您不希望某个目录被浏览, 则返回值可以
129                     //设为 Boolean.FALSE
130 }
131
132 public String getExtensionName(File f)
133 {
134     String extension = "";
135     String fileName = f.getName();
136     int index = fileName.lastIndexOf('.');
137
138     if (index > 0 && index < fileName.length()-1) {
139         extension = fileName.substring(index+1).toLowerCase();
140     }
141     return extension;
142 }
143 }

```

⊕ 说明:

- (1) 程序第 46 行, 利用 setFileView()方法设置文件类型图标。
- (2) 程序第 93~143 行, 实现 FileView 中的 5 种方法。
- (3) 方法 getName()、getDescription()与 isTraversable()返回值为 null 的话, Java 的 Look and Feel 功能会处理掉这个项目, 并取得相关值来加以设置。
- (4) 程序第 132~142 行, 在 FileIcon 类中我们增加一个 getExtensionName()方法, 用来返回文件的扩展名名称。

⊕ 程序运行结果如图 13-10 所示。



图 13-10

按下“打开文件”按钮后, 如图 13-11 所示。



图 13-11

由上面的图标可以发现，文件的类型图标改变了（.java 文件与 .class 文件分别用不同图标表示）！

13-2 建立颜色选择对话框（JColorChooer）

JColorChooser 的类层次结构图：

```
java.lang.Object
--java.awt.Component
--java.awt.Container
--javax.swing.JComponent
-- javax.swing.JColorChooser
```

Color Chooser 可以让您选择所想要的颜色，并更改某个组件的颜色。例如在画图板中，您可以在画板上画上图案，并选择各式各样的颜色来加以装饰；至于颜色的选择上，您可以在画图板中找到如下的编辑颜色对话框，如图 13-12 所示。

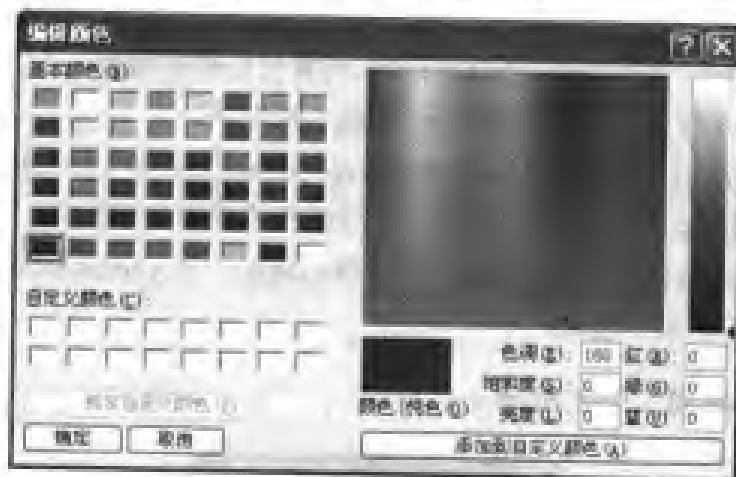


图 13-12

要做到上述的功能，在 Java 中其实是非常容易的。Java 提供了一个 JColorChooser 组件，让您很容易的输出颜色选择对话框，也很容易让您知道用户到底选择了那个颜色。表 13-3 是 JColorChooser 的构造函数：

表 13-3

JColorChooser 的构造函数：

JColorChooser()

建立一个 JColorChooser 对象，默认颜色为白色

JColorChooser(Color initialColor)

建立一个 JColorChooser 对象，并设置初始颜色

JColorChooser(ColorSelectionModel model)

以 ColorSelectionModel 构造 JColorChooser 对象

13-2-1 轻松输出颜色选择对话框

最常使用 JColorChooser 的方式是使用 JColorChooser 的静态方法（Static Method）：

showDialog()。也就是说在大部分的情况下，我们不会新建一个 JColorChooser 对象，而是直接使用 JColorChooser 的静态方法 (showDialog()) 来输出颜色选择对话框。利用这个方法我们也可以得到用户所选择的颜色，若用户没有选择则返回 null 值。

另外还有一个使用 JColorChooser 常用的方式，那就是使用 createDialog() 静态方法。使用这个静态方法后会得到一个 JDialog 对象，我们可以利用这个 JDialog 对象对颜色选择对话框做更多的设置。不过利用这个方法必须配合 JColorChooser 对象才行，也就是必须新建一个 JColorChooser 对象来。在下面范例中我们先介绍第一种最简单也是最实用的 JColorChooser 的使用方式，此程序可以让用户在 JLabel 上 Double Click 鼠标后跳出一个颜色选择对话框，选择完颜色后就能更改 JLabel 上的背景颜色了。程序如下：

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class ColorChooserDemo1 extends MouseAdapter
6  {
7      JFrame f = null;
8      JLabel label = null;
9      JLabel label1 = null;
10     JLabel label2 = null;
11     Rectangle rec1 = null;
12     Rectangle rec2 = null;
13
14     public ColorChooserDemo1()
15     {
16         f = new JFrame("ColorChooser Example");
17         Container contentPane = f.getContentPane();
18         contentPane.addMouseListener(this);
19
20         label = new JLabel(" ", JLabel.CENTER);
21         label.setPreferredSize(new Dimension(300, 20));
22
23         JPanel panel = new JPanel();
24         panel.setLayout(new GridLayout(1, 2));
25
26         label1 = new JLabel("左 Label", JLabel.CENTER);
27         label1.setBackground(Color.red);
28         label1.setForeground(Color.black);
29         label1.setOpaque(true);
30         label1.setBounds(0, 0, 150, 150);
31         panel.add(label1);
32
33         label2 = new JLabel("右 Label", JLabel.CENTER);
34         label2.setBackground(Color.green);
35         label2.setForeground(Color.black);
36         label2.setOpaque(true);
37         label2.setBounds(150, 0, 150, 150);
38         panel.add(label2);
39
40         rec1 = label1.getBounds();
41         rec2 = label2.getBounds();
42

```



```

43         contentPane.add(panel, BorderLayout.CENTER);
44         contentPane.add(label, BorderLayout.SOUTH);
45
46         f.setSize(new Dimension(300,150));
47         f.setVisible(true);
48
49         f.addWindowListener(new WindowAdapter() {
50             public void windowClosing(WindowEvent e) {
51                 System.exit(0);
52             }
53         });
54     }
55
56     public static void main(String[] arg)
57     {
58         new ColorChooserDemol();
59     }
60
61     public void mousePressed(MouseEvent e) {
62         label.setText("目前鼠标坐标 (X,Y) 为: (" + e.getX() + ", " + e.getY() + ")");
63     }
64
65     public void mouseClicked(MouseEvent e)
66     {
67         Point point = e.getPoint();
68
69         if (e.getClickCount() == 2)
70         {
71             if (rec1.contains(point))
72             {
73                 Color color = JColorChooser.showDialog(
74                     f, "Change label1 Color", Color.white);
75                 if (color != null) //若为 null 值表示用户按下 Cancel 按钮
76                     label1.setBackground(color);
77             }
78             if (rec2.contains(point))
79             {
80                 Color color = JColorChooser.showDialog(
81                     f, "Change label2 Color", Color.yellow);
82                 if (color != null) //若为 null 值表示用户按下 Cancel 按钮
83                     label2.setBackground(color);
84             }
85         }
86     }
87 }

```

◆ 说明:

- (1) 程序第 5 行, 我们继承 `MouseAdapter` 抽象类来实现处理鼠标事件的部分。
- (2) 程序第 61~86 行, 实现 `MouseAdapter` 中的 `mousePressed()` 与 `mouseClicked()` 方法。当按下鼠标时, 就能知道鼠标光标目前的位置。当连续键击鼠标两次时 (Double Click), 若光标所在位置在 label 中, 就会出现颜色选择对话框, 用户可选择任一颜色更改 label 的颜色。

- (3) 程序第 73~74 行, 利用 JColorChooser 的 showDialog() 静态方法输出颜色选择对话框, showDialog() 中的 3 个参数依次是: 对话框的父组件、颜色选择对话框标题、与对话框默认颜色。当用户选择完颜色之后, 按下“确定”按钮则返回 Color 对象, 若按下“撤消”按钮则返回 null 值。

④ 程序运行结果如图 13-13 所示。



图 13-13

对“右 Label”双击时会出现颜色选择对话框, 如图 13-14 所示。

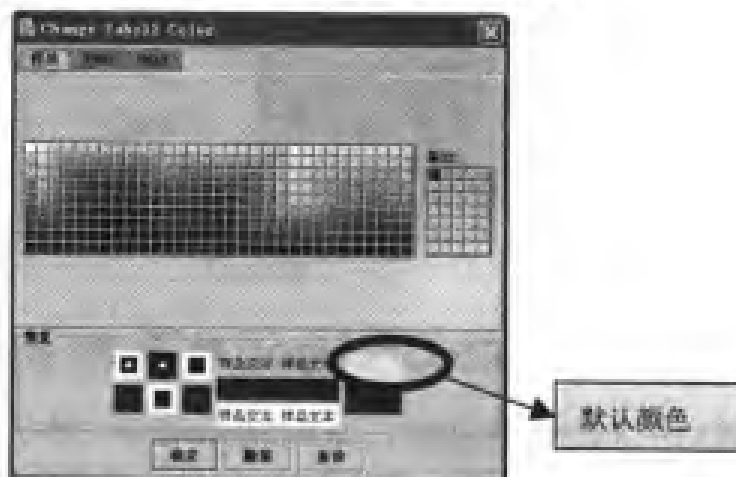


图 13-14

在颜色选择对话框中有 3 种选择颜色的模式, 分别为样品 (Swatches)、HSB 与 RGB, 可以依照您的需要选择使用哪种模式:

HSB (Hue-Saturation-Brightness) 模式, 如图 13-15 所示。



图 13-15

RGB (Red-Green-Blue) 模式, 如图 13-16 所示。



图 13-16

若按下上面的“重设”按钮会恢复成我们程序所设置的黄色。当您选择好颜色之后, 按下“确定”按钮, 原来的 label 就会变成您所设置的颜色, 如图 13-17 所示。



图 13-17

13-2-2 建立 JColorChooser 对象输出颜色选择对话框

接下来我们介绍刚刚提到的使用 JColorChooser 的第二种方式, 也就是使用 createDialog() 静态方法来输出颜色选择对话框。使用这种方式的好处是颜色选择对话框可以做出多样性的变化, 例如您可以使用 JDialog 中的 setMenuBar() 方法在颜色选择对话框中加入菜单栏, 或是利用 JDialog 的 getContentPane() 方法取得 JDialog 的 ContentPane, 然后对此 ContentPane 来做处理。下面的范例修改自上个范例程序, 并利用 createDialog() 方式来输出 JColorChooser:

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class ColorChooserDemo2 extends MouseAdapter implements
6  ActionListener
7  {
8      JFrame f = null;
9      JLabel label = null;
10     JLabel label1 = null;
11     JLabel label2 = null;
12     Rectangle rec1 = null;
13     Rectangle rec2 = null;
14     JDialog dialog = null;
15     JColorChooser colorChooser = null;
16

```

```

17     public ColorChooserDemo2()
18     {
19         f = new JFrame("ColorChooser Example");
20         Container contentPane = f.getContentPane();
21         contentPane.addMouseListener(this);
22
23         label = new JLabel(" ", JLabel.CENTER);
24         label.setPreferredSize(new Dimension(300,20));
25
26         JPanel panel = new JPanel();
27         panel.setLayout(new GridLayout(1,2));
28
29         label1 = new JLabel("左 Label",JLabel.CENTER);
30         label1.setBackground(Color.red);
31         label1.setForeground(Color.black);
32         label1.setOpaque(true);
33         label1.setBounds(0, 0, 150, 150);
34         panel.add(label1);
35
36         label2 = new JLabel("右 Label",JLabel.CENTER);
37         label2.setBackground(Color.green);
38         label2.setForeground(Color.black);
39         label2.setOpaque(true);
40         label2.setBounds(150, 0, 150, 150);
41         panel.add(label2);
42
43         rec1 = label1.getBounds();
44         rec2 = label2.getBounds();
45
46         contentPane.add(panel,BorderLayout.CENTER);
47         contentPane.add(label,BorderLayout.SOUTH);
48
49         colorChooser = new JColorChooser();
50         dialog = colorChooser.createDialog(f,        //parent component
51             "Change Color",        //title
52             true, //modal
53             colorChooser,        //JColorChooser
54             this,                //okListenr
55             this);                //cancelListener
56
57         f.setSize(new Dimension(300,150));
58         f.setVisible(true);
59
60         f.addWindowListener(new WindowAdapter() {
61             public void windowClosing(WindowEvent e) {
62                 System.exit(0);
63             }
64         });
65     }
66
67     public static void main(String[] arg)
68     {
69         new ColorChooserDemo2();
70     }
71

```

```

72     public void mousePressed(MouseEvent e) {
73         label.setText("目前鼠标坐标 (X,Y) 为: (" + e.getX() + ", " + e.getY() + ")");
74     }
75
76     boolean flag = true; //判断是要改变 label1 或是 label2 的颜色
77
78     public void mouseClicked(MouseEvent e)
79     {
80         Point point = e.getPoint();
81
82         if (e.getClickCount() == 2)
83         {
84             if (rec1.contains(point))
85             {
86                 flag = true;
87                 dialog.setTitle("Change label1 Color");
88                 dialog.show();
89             }
90             if (rec2.contains(point))
91             {
92                 flag = false;
93                 dialog.setTitle("Change label2 Color");
94                 dialog.show();
95             }
96         }
97     }
98
99     public void actionPerformed(ActionEvent ae)
100    {
101        if (ae.getActionCommand().equals("OK"))
102        {
103            if (flag == true)
104                label1.setBackground(colorChooser.getColor());
105            else
106                label2.setBackground(colorChooser.getColor());
107        }
108        if (ae.getActionCommand().equals("Cancel"))
109            dialog.dispose();
110    }
111 }

```

⊕ 说明:

- (1) 程序第 49 行, 建立一个新的 JColorChooser 对象, 默认颜色为白色。
- (2) 程序第 50~55 行, 利用 JColorChooser 的 createDialog() 静态方法取得 JDialog 对象。createDialog() 方法的最后两个参数是用来处理颜色选择对话框的“确定”与“撤销”按钮的 ActionEvent 事件。读者可发现, 这边并没有对颜色选择对话框的“重设”按钮做处理, 因为内部系统会自动处理此事件, 如此可减轻程序设计师的负担。
- (3) 程序第 84~95 行, 判断双击时鼠标所在的位置, 若在 label1 或 label2 上则设置对话框的 Title, 并用 show() 方法将此对话框显示出来。

- (4) 程序第 99~110 行, 处理用户对颜色选择对话框中的按钮事件。当用户按下“确定”按钮时, 便将 label 的背景颜色改变成用户所选择的颜色。当用户按下“撤消”按钮时, 则用 Dialog 类所提供的 dispose() 方法就可以关闭颜色选择对话框了。程序运行结果与前范例相同。

13-2-3 将 JColorChooser 置于一般容器中显示

我们刚刚所讲的 JColorChooser 都是以对话框的形式出现。事实上 JColorChooser 可以置于一般的 Java 容器上, 不过这时候您就必须实际构造出 JColorChooser 对象, 并使用 ColorSelectionModel 来管理用户所选择的颜色。ColorSelectionModel 本身是一个 Interface, 里面定义了一些用户选择颜色或设置颜色的方法, 并有 addChangeListener() 方法来检测用户是否改变了颜色的选择。要使用到 ColorSelectionModel Interface 所定义的方法, 理论上我们必须实现它, 然而 Java 本身可利用 JColorChooser 所提供的 getSelectionModel() 方法得到 ColorSelectionModel 的实体。这时候就可以直接以 ColorSelectionModel 的 addChangeListener() 方法来检测用户是否对颜色的选择有所改变, 而不需要再另外实现这些方法。

每当用户在 JColorChooser 上做一次颜色的改变时, 就会触发 ChangeEvent 事件, 因此我们必须实现 ChangeListener 接口来处理这个事件。此接口只定义了一个方法, 那就是 stateChanged()。在下面的范例中我们将 JColorChooser 放在 JPanel 上, 并在 label 上输出用户所选择的颜色与颜色参数。

程序如下:

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import javax.swing.colorchooser.*;
5  import javax.swing.event.*;
6
7  public class ColorChooserDemo3 implements ChangeListener
8  {
9      JFrame f = null;
10     JLabel label = null;
11     JColorChooser colorChooser = null;
12
13     public ColorChooserDemo3()
14     {
15         f = new JFrame("ColorChooser Example");
16         Container contentPane = f.getContentPane();
17
18         label = new JLabel(" ", JLabel.CENTER);
19         //设置 label 背景颜色为不透明, 这样才可以将 label 的背景颜色显示出来
20         label.setOpaque(true);
21         //设置 label 上字体的颜色, 也就是 label 的前景颜色
22         label.setForeground(Color.black);
23
24         JPanel panel = new JPanel();
25         colorChooser = new JColorChooser();
26         panel.add(colorChooser);
27         ColorSelectionModel selectModel =
28         colorChooser.getSelectionModel();
29         selectModel.addChangeListener(this);
30

```

```

31     contentPane.add(label, BorderLayout.NORTH);
32     contentPane.add(panel, BorderLayout.CENTER);
33
34     f.pack();
35     f.setVisible(true);
36
37     f.addWindowListener(new WindowAdapter() {
38         public void windowClosing(WindowEvent e) {
39             System.exit(0);
40         }
41     });
42 }
43
44 public static void main(String[] arg)
45 {
46     new ColorChooserDemo3();
47 }
48
49 public void stateChanged(ChangeEvent e)
50 {
51     Color color = colorChooser.getColor();
52     label.setBackground(color);
53     label.setText("您选择的颜色参数为: R: "+color.getRed()+
54                 " G: "+color.getGreen()+" B: "+color.getBlue());
55 }
56 }

```

说明:

- (1) ChangeEvent 是 Swing 的事件，因此在程序第 5 行中，我们必须将 Swing 的 event package import 进来。
- (2) 程序第 25~29 行，建立一个 JColorChooser 对象，并以 getSelectionModel() 方法取得 ColorSelectionModel 实体，并在这个实体上以 addChangeListener() 方法检测用户是否有改变颜色。
- (3) 程序第 49~55 行，实现 ChangeListener Interface，stateChanged() 方法是在处理 ChangeEvent 事件。当用户改变颜色的选择时，我们就在 label 上输出用户所选择的颜色，并显示颜色参数。

程序运行结果如图 13-18 所示。

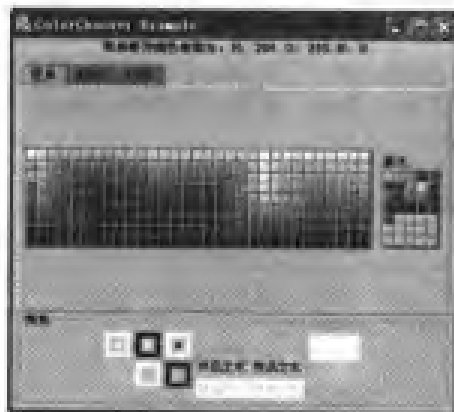


图 13-18

13-2-4 改变 JColorChooser 的颜色选择面版

在以前我们曾经提到 JColorChooser 提供 3 种面版让我们选择, 分别是“Swatches”、“HSB”与“RGB”。这 3 个面版是 Java 已经构造好的颜色面版, 如果您要自行产生自己所设计的颜色面版, 您可以继承 AbstractColorChooserPanel 这个抽象类, 并实现里面的抽象方法, 这个抽象类是位于 javax.swing.colorchooser 这个 package 中。当您实现完成之后, 您可以使用 JColorChooser 类提供的 addChooserPanel() 方法, 或是 setChooserPanels() 方法, 将您所设计的颜色面版增加到上图的颜色显示面版中。

若您想删除某一个颜色面版模式, 您可以先使用 JColorChooser 类提供的 getChooserPanels() 方法, 得到类型为 AbstractColorChooserPanel 的 Array Object。例如在上图中, “Swatches” 就会放在此类型的 Array[0] 中, 而 “HSB” 会放在 Array[1] 中, 以此类推。得到此 Array Object 之后, 我们就可以利用 JColorChooser 类提供的 removeChooserPanel() 方法, 决定删除哪一个颜色面版。例如我们若要删除上图的 “HSB” 面版, 我们可以在上面范例的第 25 行下面增加这 2 行程序:

```
AbstractColorChooserPanel[] colorPanel = colorChooser.getChooserPanels();
colorChooser.removeChooserPanel(colorPanel[1]);
```

⊕ 则运行结果将如图 13-19 所示。

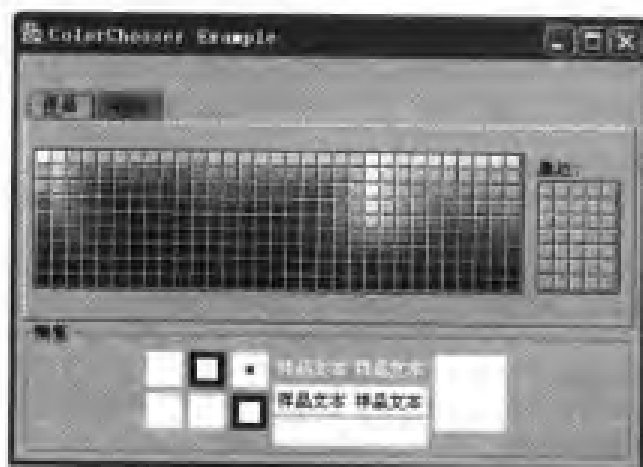


图 13-19

读者可自行试试自定义颜色面版的功能。

13-3 建立分隔线 (JSeparator)

JSeparator 的类层次结构图:

```
java.lang.Object
--java.awt.Component
--java.awt.Container
--javax.swing.JComponent
-- javax.swing.JSeparator
```


JSeparator 通常是用在菜单 (Menu) 或工具栏 (ToolBar) 上, 可以明显地分隔出不同的功能区域。在 JMenu 或 JPopupMenu 中我们可以使用 addSeparator() 方法轻易的加入分隔线, 但若是是在一般的面版中呢? 这时候我们必须自行建立 JSeparator 对象, 然后再依照需要将分隔线放在所想要的位置上。JSeparator 有水平与垂直两种, 建立的方式非常简单, 我们先来看 JSeparator 的构造函数, 如表 13-4 所示。

表 13-4

JSeparator 的构造函数

JSeparator()
建立水平的 JSeparator 组件
JSeparator(int orientation)
建立水平或垂直的 JSeparator 组件

JSeparator 类所提供的方法跟其他 Swing 组件比较起来算是少了许多, 因为分隔线本身并没有什么功能可言, 主要是设置分隔线的方向, 其他如分隔线的长短或设置位置的方法, 都可以在它的父类 JComponent 中找到。在下面的范例中我们将简单介绍如何使用 JSeparator, 并显示出水平与垂直分隔线的外观:

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class SeparatorDemol
6  {
7      JFrame f = null;
8
9      public SeparatorDemol()
10     {
11         f = new JFrame("Separator Example");
12         Container contentPane = f.getContentPane();
13         contentPane.setLayout(new BorderLayout(1,2));
14
15         JPanel panell = new JPanel(new GridLayout(2,1));
16         JLabel label = new JLabel("水平分隔线",JLabel.CENTER);
17         JSeparator seph = new JSeparator();
18         panell.add(label);
19         panell.add(seph);
20
21         JTextArea textarea = new JTextArea();
22         textarea.setPreferredSize(new Dimension(150,100));
23         JPanel panel2 = new JPanel(new BorderLayout());
24         panel2.add(panell,BorderLayout.NORTH);
25         panel2.add(textarea,BorderLayout.CENTER);
26
27         JPanel panel3 = new JPanel(new GridLayout(1,3));
28         label = new JLabel("垂直");
29         label.setVerticalAlignment(JLabel.CENTER);
30         panel3.add(label);
31         JSeparator sepv = new JSeparator();
32         sepv.setOrientation(JSeparator.VERTICAL);
33         panel3.add(sepv);
```

```

34
35     contentPane.add(panel2,BorderLayout.CENTER);
36     contentPane.add(panel3,BorderLayout.EAST);
37     f.pack();
38     f.setVisible(true);
39
40     f.addWindowListener(new WindowAdapter() {
41         public void windowClosing(WindowEvent e) {
42             System.exit(0);
43         }
44     });
45 }
46
47 public static void main(String[] arg)
48 {
49     new SeparatorDemo1();
50 }
51 }

```

说明：

- (1) 在此程序中，JFrame 的 Content Pane 为 BorderLayout，我们在 BorderLayout 的中间放置一个 JLabel，一个水平分隔线，一个 JTextArea，而右边放置一个 JLabel 与一个垂直分隔线。
- (2) 程序第 17 行，建立一个水平的分隔线。
- (3) 程序第 31~32 行，建立一个垂直的分隔线。

程序运行结果如图 13-20 所示。



图 13-20

13-4 本章总结

在本章中我们详细地介绍了文件选择对话框的各种功能，包括基本文件对话框的建置、文件类型的过滤、显示个别的文件图标等等。后面我们介绍了编辑颜色对话框的功能，让用户可以轻松地为自己的组件选择所喜欢的颜色，并让用户知道如何将颜色选择面板置于一般

容器中，且可以删除 Java 提供的颜色选择面版，或是增加自定义的颜色面版。在本章最后，我们简单地介绍了 Java 分隔线的使用，让用户更能了解到分隔线的功能与作用。

13-5 本章习题

1. 试设计一个可以过滤出*.doc 类型的文件选择对话框，并将图标更改成 Office Word 的图标。
2. 修改 FileChooserDemo1.java 文件，使存盘时若文件已经存在，则提示用户文件已存在，并显示是否覆盖、附加（Append），或是取消。
3. 简易设计一个具有画图功能的绘图小软件，并利用 JColorChooser 改变绘图组件上的颜色。
4. 试继承 AbstractColorChooserPanel 抽象类，构造一个自行设计的颜色选择面版。
5. 试述 Java 提供了哪 3 种选择颜色的模式。在显示时可否删除其中一种？如何删除？

14

滑动杆(Slider)、时间控制(Timer)、 进度组件(Progress)的使用与介绍

在本章中，我们将详细地介绍与 Progress Bar 相关的组件。包括 JSlider、Timer、JProgressBar、ProgressMonitor 与 ProgressMonitorInputStream。JSlider 如同电器用品上机械式的微调控制杆，可以让用户拖曳滑动杆来微调程序的运行。Timer 可以控制一种周期性的变化事件，如更换图片或定时更新数据。JProgressBar 可以显示目前的进度状况，让用户清楚知道系统的运行信息。ProgressMonitor 与 ProgressMonitorInputStream 可以用另一个对话框来显示进度状况，并可设置显示时机，这两个组件在使用的时机上有些许的不同。读者在阅读完本章后就可以写一个漂亮的显示图程序了。

14-1 使用 JSlider 组件

JSlider 的类层次结构图:

```
java.lang.Object
--java.awt.Component
  --java.awt.Container
    --javax.swing.JComponent
      -- javax.swing.JSlider
```

Slider 就好像电器用品上机械式的微调控制杆, 例如收音机上的调频扭, 或是汽车上的冷气温度调节杆 (现在汽车越做越精致, 想要在汽车面版上找到机械式的组件好像不是那么容易了), 利用这些滑动杆我们就能控制相关的系统, 达到我们想要的效果。我们曾在第 5 章中提到 JScrollBar 这个组件, 可以让用户决定拉曳时一次滚动的区域大小, 并且可以得到当前滚动条杆上的值, 看起来 JScrollBar 好像可以当作这种微调的组件。然而 JScrollBar 通常置于窗口的最右边或最下面, 且常搭配 JScrollPane 来使用, 因此在外观上或实际应用上并不适合当微调工具使用。不过不用担心, Java 提供了一个特别为微调设计的组件, 那就是 JSlider。JSlider 不仅可以置于面版的任何地方, 也可以在 JSlider 上标上刻度与数字, 既美观又实用, 我们现在就来看如何使用 JSlider 吧。表 14-1 是 JSlider 的构造函数。

表 14-1

JSlider 构造函数
JSlider() 建立一个水平的 JSlider 对象, 刻度值从 0~100, 初始刻度值为 50
JSlider(BoundedRangeModel brm) 使用默认模式建立一个水平的 JSlider 对象
JSlider(int orientation) 建立一个自定义方向的 JSlider 对象, 刻度值从 0~100, 初始刻度值为 50
JSlider(int min, int max) 建立一个水平的 JSlider 对象, 自定义刻度值, 从 min~max, 初始刻度值为 50
JSlider(int min, int max, int value) 建立一个水平的 JSlider 对象, 自定义刻度与刻度初始值
JSlider(int orientation, int min, int max, int value) 建立一自定义方向、刻度与刻度初始值的 JSlider 对象

要使用 JSlider 组件就不得不提到 JSlider 事件的处理。当用户在 JSlider 上移动滑动杆时, 就会产生 ChangeEvent 事件, 若我们要处理 ChangeEvent 事件就必须实现 ChangeListener 接口, 此接口定义了一种方法, 那就是 stateChanged()。通常我们在这个方法上会取得或设置滑动杆的相关信息, 例如滑动杆的延伸区 (extent)、最大最小值或滑动杆目前所在的刻度等等。我们来看下一节所举的范例。

14-1-1 建立 JSlider 组件

在这个范例中我们建立了 3 个 JSlider 组件, 并对每个 JSlider 组件做相关的设置, 例如

设置方向、初始值、最大最小值、延伸区值 (Extent) 等。程序如下:

范例 SliderDemo1.java (文件位于随书光盘目录 examch14\SliderDemo1.java)

```

1  import javax.swing.*;
2  import javax.swing.border.*;
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.event.*;
6
7  public class SliderDemo1 implements ChangeListener
8  {
9      JFrame f = null;
10     JSlider slider1;
11     JSlider slider2;
12     JSlider slider3;
13     JLabel label1;
14     JLabel label2;
15     JLabel label3;
16
17     public SliderDemo1()
18     {
19         f = new JFrame("JSlider Example");
20         Container contentPane = f.getContentPane();
21
22         JPanel panel1 = new JPanel();
23         panel1.setLayout(new GridLayout(2,1));
24         slider1 = new JSlider();
25         label1 = new JLabel("目前刻度: "+slider1.getValue());
26         panel1.add(label1);
27         panel1.add(slider1);
28         panel1.setBorder(BorderFactory.createTitledBorder(
29             BorderFactory.createEtchedBorder(),"Slider 1",TitledBorder.LEFT,
30             TitledBorder.TOP));
31
32         JPanel panel2 = new JPanel();
33         panel2.setLayout(new GridLayout(2,1));
34         slider2 = new JSlider(JSlider.HORIZONTAL);
35         slider2.setMinimum(0);
36         slider2.setMaximum(100);
37         slider2.setValue(30);
38         slider2.setExtent(50);
39         label2 = new JLabel("目前刻度: "+slider2.getValue());
40         panel2.add(label2);
41         panel2.add(slider2);
42         panel2.setBorder(BorderFactory.createTitledBorder(
43             BorderFactory.createEtchedBorder(),"Slider 2",TitledBorder.LEFT,
44             TitledBorder.TOP));
45
46         JPanel panel3 = new JPanel();
47         panel3.setLayout(new GridLayout(2,1));
48         slider3 = new JSlider(20,80);
49         slider3.setOrientation(JSlider.VERTICAL);
50         label3 = new JLabel("目前刻度: "+slider3.getValue());

```

```

51     panel3.add(label3);
52     panel3.add(slider3);
53     panel3.setBorder(BorderFactory.createTitledBorder(
54     BorderFactory.createEtchedBorder(),"Slider 3",TitledBorder.LEFT,
55     TitledBorder.TOP));
56
57     slider1.addChangeListener(this);
58     slider2.addChangeListener(this);
59     slider3.addChangeListener(this);
60
61     panell1.setPreferredSize(new Dimension(300,100));
62     panel2.setPreferredSize(new Dimension(300,100));
63     panel3.setPreferredSize(new Dimension(150,200));
64
65     GridBagConstraints c;
66     int gridx,gridy,gridwidth,
67         gridheight,anchor,fill,ipadx,ipady;
68     double weightx,weighty;
69     Insets inset;
70
71     GridBagLayout gridbag = new GridBagLayout();
72     contentPane.setLayout(gridbag);
73
74     gridx=0;           //第 0 行
75     gridy=0;           //第 0 列
76     gridwidth = 2;      //占两单位宽度
77     gridheight = 1;     //占一单位高度
78     weightx = 0;        //窗口增大时组件宽度增大比率 0
79     weighty = 0;        //窗口增大时组件高度增大比率 0
80     anchor = GridBagConstraints.CENTER; //容器大于组件 size 时将组件
81 //置于容器中央
82     fill = GridBagConstraints.BOTH;      //窗口拉大会填满水平与垂
83 //直空间
84     inset = new Insets(0,0,0,0);        //组件间距
85     ipadx = 0;                          //组件内水平宽度
86     ipady = 0;                          //组件内垂直高度
87     c = new GridBagConstraints(gridx,gridy,gridwidth,gridheight,
88         weightx,weighty,anchor,fill,inset,ipadx,ipady);
89     gridbag.setConstraints(panell1,c);
90     contentPane.add(panell1);
91
92     gridx=0;
93     gridy=1;
94     c = new GridBagConstraints(gridx,gridy,gridwidth,gridheight,
95         weightx,weighty,anchor,fill,inset,ipadx,ipady);
96     gridbag.setConstraints(panel2,c);
97     contentPane.add(panel2);
98
99     gridx=2;
100    gridy=0;
101    gridwidth = 1;      //占一单位宽度
102    gridheight = 2;     //占两单位高度
103    c = new GridBagConstraints(gridx,gridy,gridwidth,gridheight,

```

```

104         weightx,weighty,anchor,fill,inset,ipadx,ipady);
105     gridbag.setConstraints(panel3,c);
106     contentPane.add(panel3);
107
108     f.pack();
109     f.setVisible(true);
110
111     f.addWindowListener(new WindowAdapter() {
112         public void windowClosing(WindowEvent e) {
113             System.exit(0);
114         }
115     });
116 }
117
118 public static void main(String[] args)
119 {
120     new SliderDemol();
121 }
122
123 public void stateChanged(ChangeEvent e)
124 {
125     if ((JSlider)e.getSource() == slider1)
126         label1.setText("目前刻度: "+slider1.getValue());
127     if ((JSlider)e.getSource() == slider2)
128         label2.setText("目前刻度: "+slider2.getValue());
129     if ((JSlider)e.getSource() == slider3)
130         label3.setText("目前刻度: "+slider3.getValue());
131 }
132 }

```

说明:

- (1) 程序第 5 行,ChangeEvent 事件是属于 Swing 的事件,若要处理此事件必须将 import Swing 的 event package 进来。
- (2) 程序第 24 行,建立一个默认的 JSlider 组件。
- (3) 程序第 34~38 行,建立一个水平方向的 JSlider 组件,并设置其最大值、最小值、初始值与延伸区值。所谓的延伸区值我们在第 5 章的 JScrollBar 中也提到过,意思是限制 JSlider 刻度可变动的范围,也就是说延伸区就像是一个障碍区,是无法通行的。延伸区设得越大,刻度可变动的范围就越小。例如,若 minimum 值设为 0, maximum 值设为 100,而 extent 值设为 0,则 JSlider 刻度可变动的区域大小为 $100-0-0=100$ 个刻度(从 0~100)。在此例中,我们设置 minimum 值设为 0, maximum 值设为 100,而 extent 值设为 50,因此 JSlider 刻度可变动的区域大小为 $100-50-0=50$ 个刻度(从 0~50)。
- (4) 程序第 48~49 行,建立一个具有最大最小值的 JSlider 组件,并设置此 JSlider 组件为垂直方向。
- (5) 程序第 123~131 行,处理 ChangeEvent 事件,当用户移动滑动杆时, label 上的值会随着用户的移动而改变。

◆ 程序运行结果如图 14-1 所示。



图 14-1

由于 slider2 设置延伸区值 (Extent) 为 50, 因此 slider2 刻度可变动的区域只到 50, 因此当 slider2 超过 50 刻度时就算您再向右移动, 刻度值一样维持在 50 上, 如图 14-2 所示。

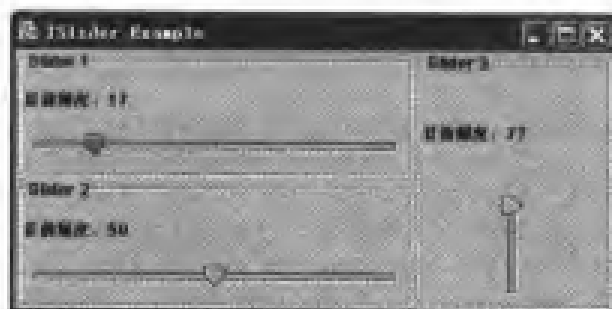


图 14-2

14-1-2 为 JSlider 组件加入刻度

上面这个例子中, 您是否觉得好像少了什么? 没错, 就是刻度! 接下来, 我们来设置 JSlider 的刻度与数字, 并增加一些设置项目, 完整的 JSlider 组件就大功告成了! 程序如下:

范例 SliderDemo2.java (文件位于随书光盘目录 exam\ch14\SliderDemo2.java)

```
1  import javax.swing.*;
2  import javax.swing.border.*;
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.event.*;
6
7  public class SliderDemo2 implements ChangeListener
8  {
9      JFrame f = null;
10     JSlider slider1;
11     JSlider slider2;
12     JSlider slider3;
13     JLabel label1;
14     JLabel label2;
15     JLabel label3;
16
```

```

17     public SliderDemo2()
18     {
19         f = new JFrame("JSlider Example");
20         Container contentPane = f.getContentPane();
21
22         JPanel panel1 = new JPanel();
23         panel1.setLayout(new GridLayout(2,1));
24         slider1 = new JSlider();
25         slider1.setPaintTicks(true);
26         slider1.setMajorTickSpacing(20);
27         slider1.setMinorTickSpacing(10);
28         slider1.setPaintLabels(true);
29         slider1.setPaintTrack(true);
30         slider1.setSnapToTicks(true);
31         label1 = new JLabel("目前刻度: "+slider1.getValue());
32         panel1.add(label1);
33         panel1.add(slider1);
34         panel1.setBorder(BorderFactory.createTitledBorder(
35             BorderFactory.createEtchedBorder(),"Slider 1",TitledBorder.LEFT,
36             TitledBorder.TOP));
37
38         JPanel panel2 = new JPanel();
39         panel2.setLayout(new GridLayout(2,1));
40         slider2 = new JSlider(JSlider.HORIZONTAL);
41         slider2.setMinimum(0);
42         slider2.setMaximum(100);
43         slider2.setValue(30);
44         slider2.setExtent(50);
45         slider2.setPaintTicks(true);
46         slider2.setMajorTickSpacing(10);
47         slider2.setMinorTickSpacing(2);
48         slider2.setPaintLabels(true);
49         slider2.putClientProperty("JSlider.isFilled",Boolean.TRUE);
50         label2 = new JLabel("目前刻度: "+slider2.getValue());
51         panel2.add(label2);
52         panel2.add(slider2);
53         panel2.setBorder(BorderFactory.createTitledBorder(
54             BorderFactory.createEtchedBorder(),"Slider 2",TitledBorder.LEFT,
55             TitledBorder.TOP));
56
57         JPanel panel3 = new JPanel();
58         panel3.setLayout(new GridLayout(2,1));
59         slider3 = new JSlider(20,80);
60         slider3.setOrientation(JSlider.VERTICAL);
61         slider3.setPaintTicks(true);
62         slider3.setMajorTickSpacing(30);
63         slider3.setMinorTickSpacing(10);
64         slider3.setPaintLabels(true);
65         slider3.putClientProperty("JSlider.isFilled",Boolean.TRUE);
66         label3 = new JLabel("目前刻度: "+slider3.getValue());
67         panel3.add(label3);
68         panel3.add(slider3);
69         panel3.setBorder(BorderFactory.createTitledBorder(
70             BorderFactory.createEtchedBorder(),"Slider 3",TitledBorder.LEFT,
71             TitledBorder.TOP));

```

```

72
73     slider1.addChangeListener(this);
74     slider2.addChangeListener(this);
75     slider3.addChangeListener(this);
76
77     panel1.setPreferredSize(new Dimension(300,130));
78     panel2.setPreferredSize(new Dimension(300,130));
79     panel3.setPreferredSize(new Dimension(150,260));
80
81     GridBagConstraints c;
82     int gridx,gridy,gridwidth,
83         gridheight,anchor,fill,ipadx,ipady;
84     double weightx,weighty;
85     Insets inset;
86     .....
87     .....
88     下面程序代码与上例相同

```

⊕ 说明:

- (1) 程序第 25 行, `setPaintTicks()` 方法设置是否在 `JSlider` 上加上刻度, 若设为 `true`, 则程序第 26~27 行才会起作用。
- (2) 程序第 26~27 行, 设置大刻度与小刻度之间的距离 (`setMajorTickSpacing()` 与 `setMinorTickSpacing()` 方法)。例如若大刻度间距离为 30, 小刻度间距离为 10, 则表示 2 个大刻度间会有 3 个小刻度。
- (3) 程序第 28 行, `setPaintLabels()` 方法为设置是否绘制数字标记。若设为 `true`, 则 `JSlider` 刻度上就会有数值出现。
- (4) 程序第 29 行, `setPaintTrack()` 方法表示是否出现滑动杆的横杆。默认值为 `true`。
- (5) 程序第 30 行, `setSnapToTicks()` 方法表示一次移动一个小刻度, 而不再是一次移动一个单位刻度。
- (6) 程序第 49 行, `JComponent` 提供一个 `putClientProperty()` 方法, 可以使得 `JSlider` 中小于滑动杆位置与大于滑动杆位置间的颜色不一样, 这样的视觉效果会比较好。
“`JSlider.isFilled`”值是定义在 `MetalSliderUI` 这个类的 `SLIDER_FILL` 常数中, 此类是定义 Java 的 `JSlider` 默认外观, 您可以在 `javax.swing.plaf.metal` package 中找到这个类。

⊕ 程序运行结果如图 14-3 所示。



图 14-3

由于我们在 slider1 中设置 setSnapToTicks() 为 true, 因此滑动杆一次移动一小刻度的距离, 也就是 10 个刻度, 而不是一个单位刻度。读者可自行试试看。

若我们将程序第 25 行的 setPaintTicks() 设为 false, 则 slider1 变成如图 14-4 所示。

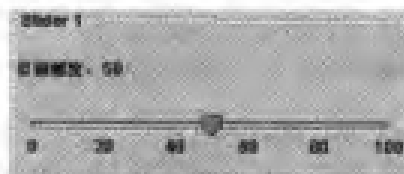


图 14-4

若我们将程序第 28 行的 setPaintLabels() 设为 false, 则 slider1 变成如图 14-5 所示。

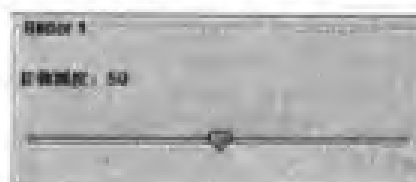


图 14-5

若我们将程序第 29 行的 setPaintTrack() 设为 false, 则 slider1 变成如图 14-6 所示。

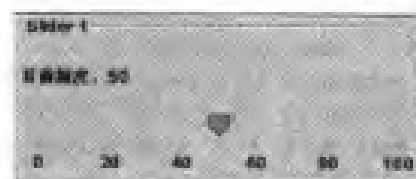


图 14-6

14-1-3 自定义 JSlider 标记名称

我们在上个范例中看到的 JSlider 标记都是数字形态, 当然我们也可以自定义想要的标记文字, 这时候我们就必须使用 JSlider 所提供的 setLabelTable() 方法了。我们来看下面的例子:

范例 SliderDemo3.java (文件位于随书光盘目录 exam\ch14\SliderDemo3.java)

```
1  import javax.swing.*;
2  import javax.swing.border.*;
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.event.*;
6  import java.util.*;
7
8  public class SliderDemo3 implements ChangeListener
9  {
10     JFrame f = null;
11     JSlider slider1;
12     JSlider slider2;
13     JSlider slider3;
14     JLabel label1;
```

```
15     JLabel label2;
16     JLabel label3;
17
18     public SliderDemo3()
19     {
20         f = new JFrame("JSlider Example");
21         Container contentPane = f.getContentPane();
22
23         JPanel panel1 = new JPanel();
24         panel1.setLayout(new GridLayout(2,1));
25         slider1 = new JSlider();
26         slider1.setPaintTicks(true);
27         slider1.setMajorTickSpacing(20);
28         slider1.setMinorTickSpacing(10);
29         slider1.setPaintLabels(true);
30         slider1.setPaintTrack(true);
31         slider1.setSnapToTicks(true);
32         label1 = new JLabel("目前刻度: "+slider1.getValue());
33         panel1.add(label1);
34         panel1.add(slider1);
35         panel1.setBorder(BorderFactory.createTitledBorder(
36             BorderFactory.createEtchedBorder(),"Slider 1",TitledBorder.LEFT,
37             TitledBorder.TOP));
38
39         Hashtable table = new Hashtable();
40         table.put(new Integer( 0 ),new JLabel("低"));
41         table.put(new Integer( 50 ),new JLabel("中"));
42         table.put(new Integer( 100 ),new JLabel("高"));
43         slider1.setLabelTable(table);
44
45         JPanel panel2 = new JPanel();
46         panel2.setLayout(new GridLayout(2,1));
47         slider2 = new JSlider(JSlider.HORIZONTAL);
48         slider2.setMinimum(0);
49         slider2.setMaximum(100);
50         slider2.setValue(30);
51         slider2.setExtent(50);
52         slider2.setPaintTicks(true);
53         slider2.setMajorTickSpacing(10);
54         slider2.setMinorTickSpacing(5);
55         slider2.setPaintLabels(true);
56         slider2.putClientProperty("JSlider.isFilled",Boolean.TRUE);
57         label2 = new JLabel("目前刻度: "+slider2.getValue());
58         panel2.add(label2);
59         panel2.add(slider2);
60         panel2.setBorder(BorderFactory.createTitledBorder(
61             BorderFactory.createEtchedBorder(),"Slider 2",TitledBorder.LEFT,
62             TitledBorder.TOP));
63
64         table = new Hashtable();
65         table.put(new Integer( 0 ),new JLabel("弱"));
66         table.put(new Integer( 25 ),new JLabel("有点弱"));
67         table.put(new Integer( 50 ),new JLabel("中"));
68         table.put(new Integer( 75 ),new JLabel("有点强"));
69         table.put(new Integer( 100 ),new JLabel("强"));
```

```

70         slider2.setLabelTable(table);
71
72         JPanel panel3 = new JPanel();
73         panel3.setLayout(new GridLayout(2,1));
74         .....
75         .....
76         下面程序代码与上例相同

```

说明：

- (1) 要改变标记值必须使用 `setLabelTable(Dictionary table)` 方法，参数 `Dictionary` 本身是一个抽象类，我们并不能直接就新建一个 `Dictionary` 对象，而必须使用它的子类 `Hashtable` 来产生 `Dictionary` 类型的对象。`Hashtable` 存储信息的方式是以 `key-value pair` 类型来存储，换句话说，当您要找某个对象时，您就必须知道此对象的 `key` 值。而在此例中，我们要更换原有 `JSlider` 上的文字，必须指明那个数字要更改成什么文字。例如 `table.put(new Integer(0), new JLabel("弱"))` 就表示数字 0 要变更成文字“弱”，在 `Hashtable` 中，`Integer(0)` 就是对象 `JLabel("弱")` 的 `key` 值。

程序运行结果如图 14-7 所示。



图 14-7

14-2 使用 Timer 组件

```

java.lang.Object
-- javax.swing.Timer

```

使用 `Timer` 组件可以让您在一段时间内依次做出您指定的操作，这在动画的展示上非常有用。如果您用过如 `ACDSee` 这类的看图软件，您可以发现这类软件都会提供一种功能，那就是自动换图的功能，而且也可以让您设置换图时间间隔的长短。在 `Java` 中，`Swing` 的 `Timer` 组件就可以让您做到这样的功能，而且非常容易使用，下面我们先来看 `Timer` 的构造函数，如表 14-2 所示。

表 14-2

Timer 构造函数
<code>Timer(int delay, ActionListener listener)</code> 建立一个 <code>Timer</code> 组件，并在每一次 <code>delay</code> 的时间点上触发 <code>ActionEvent</code>

使用 Timer 组件它会根据您所给予的 delay 时间, 周期性的触发 ActionEvent 事件, 如果您要处理这个事件, 您必须实现 ActionListener 接口所定义的 actionPerformed() 方法。要开始激活 Timer 组件, 您可以用 start() 方法, 要停止 Timer 组件可以使用 stop() 方法, 要重新激活 Timer 组件可以使用 restart() 方法, 若想 Timer 组件只触发一次 ActionEvent 事件, 可利用 setRepeats(false) 方法, 将参数设为 false; 若要设置 delay 时间则可用 setDelay() 方法。

事实上使用 Timer 组件表示在程序背后是利用 Threads 在运行 Timer 的工作, 因此您也可以利用 Thread 的功能来自行制造出这样的效果, 不过这不在本书的讨论范围内。在下面的范例中我们实作一个显示图程序, 可让用户调整显示图时间的快慢。

范例 TimerDemo1.java (文件位于随书光盘目录 examch14\TimerDemo1.java)

```

1  import javax.swing.*;
2  import javax.swing.border.*;
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.event.*;
6  import java.util.*;
7
8  public class TimerDemo1 implements ActionListener,ChangeListener
9  {
10     JFrame f = null;
11     ImageIcon[] icons;
12     JSlider slider1;
13     JLabel label;
14     JToggleButton toggleb1,toggleb2;
15     JButton b;
16     javax.swing.Timer timer;
17     int index = 0;
18
19     public TimerDemo1()
20     {
21         f = new JFrame("Timer Example");
22         Container contentPane = f.getContentPane();
23         icons = new ImageIcon[5];
24         for (int i=0 ; i<5 ; i++)
25             icons[i] = new ImageIcon(".\\icons\\"+(i+1)+".jpg");
26
27         label = new JLabel(icons[0]);
28         JPanel panell = new JPanel();
29         panell.setLayout(new GridLayout(2,1));
30         slider1 = new JSlider();
31         slider1.setPaintTicks(true);
32         slider1.setMajorTickSpacing(20);
33         slider1.setMinorTickSpacing(10);
34         slider1.setPaintLabels(true);
35         slider1.addChangeListener(this);
36         panell.add(slider1);
37
38         JPanel buttonPanel = new JPanel();
39         buttonPanel.setLayout(new GridLayout(1,3));
40         toggleb1 = new JToggleButton("Start");
41         toggleb1.addActionListener(this);

```

```

42         buttonPanel.add(toggleb1);
43         b = new JButton("Restart");
44         b.addActionListener(this);
45         buttonPanel.add(b);
46         toggleb2 = new JToggleButton("Don't Repeat");
47         toggleb2.addActionListener(this);
48         buttonPanel.add(toggleb2);
49         panell.add(buttonPanel);
50
51         Hashtable table = new Hashtable();
52         table.put(new Integer( 0 ),new JLabel("快"));
53         table.put(new Integer( 50 ),new JLabel("中"));
54         table.put(new Integer( 100 ),new JLabel("慢"));
55         slider1.setLabelTable(table);
56
57         timer = new javax.swing.Timer(slider1.getValue()*10,this);
58
59         contentPane.add(label,BorderLayout.CENTER);
60         contentPane.add(panell,BorderLayout.SOUTH);
61
62         f.pack();
63         f.setVisible(true);
64
65         f.addWindowListener(new WindowAdapter() {
66             public void windowClosing(WindowEvent e) {
67                 System.exit(0);
68             }
69         });
70     }
71
72     public static void main(String[] args)
73     {
74         new TimerDemol();
75     }
76
77     public void actionPerformed(ActionEvent e)
78     {
79         if (e.getSource() == toggleb1)
80         {
81             if (e.getActionCommand().equals("Start"))
82             {
83                 timer.start();
84                 toggleb1.setText("Stop");
85             }
86             if (e.getActionCommand().equals("Stop"))
87             {
88                 timer.stop();
89                 toggleb1.setText("Start");
90             }
91         }
92         if (e.getSource() == toggleb2)
93         {
94             if(timer.isRepeats())
95             {
96                 timer.setRepeats(false);

```



```

97         }
98     else
99     {
100         timer.setRepeats(true);
101         timer.start();
102     }
103 }
104 if (e.getSource() == b)
105 {
106     slider1.setValue(50);
107     timer.restart();
108 }
109
110 if (e.getSource() == timer)
111 {
112     if (index == 5)
113         index = 0;
114     label.setIcon(icons[index]);
115     label.repaint();
116     //f.pack(); //若要窗口随图形大小变化,可加入此行
117     index++;
118 }
119 }
120
121 public void stateChanged(ChangeEvent e1)
122 {
123     timer.setDelay(slider1.getValue()*10);
124 }
125 }

```

⊕ 说明:

- (1) 程序第 57 行中, 由于 Java 的 Timer 组件有两种, 一种是 javax.swing.Timer, 一种是 java.util.Timer, 若我们在程序中 import 了这两种 package, 则系统将不知道到底要产生哪种 Timer 组件, 就如同本范例一般。因此我们必须在新建 Timer 组件的同时, 指定要新建出哪一种类型的 Timer 组件, 在此我们当然是要产生 Swing 的 Timer 组件。
- (2) 程序第 77~119 行, 处理按钮事件与 Timer 事件。
- (3) 程序第 79~91 行, 当用户按下“Start”按钮时, Timer 开始运行, 且“Start”按钮会变成“Stop”按钮, 若用户再次按下“Stop”按钮, 则 Timer 暂停运行, 且“Stop”按钮变成“Start”按钮。
- (4) 程序第 92~103 行, 当用户按下“Don't Repeat”按钮时, 则 Timer 事件只触发一次, 若再按一次“Don't Repeat”按钮, 则 Timer 继续运行。
- (5) 程序第 104~108 行, 当用户按下“Restart”按钮时, 则 Timer 组件的 delay 值恢复成初始值, 并重新运行 Timer。
- (6) 程序第 110~118 行, 处理 Timer 所产生的(ActionEvent)事件。每次时间一到 delay 所设置的时间, label 上的图片就会更换一次。

(7) 程序第 121~124 行, 处理 slider1 所产生的 ChangeEvent 事件。当用户移动 slider1 的滑动杆时, 等于重新设置 Timer 的 delay 时间。

④ 程序运行结果如图 14-8 所示。

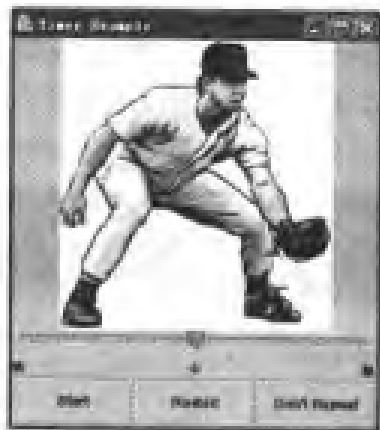


图 14-8

按下“Start”按钮后, Timer 开始运行, 且“Start”按钮会变成“Stop”按钮, 如图 14-9 所示。

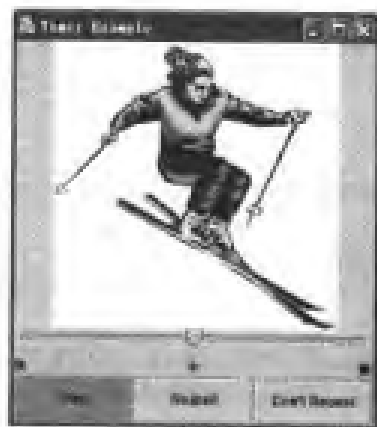


图 14-9

读者可自行试试这个程序的运行效果。

14-3 使用 Progress Bar 组件

JProgressBar 的类层次结构图:

```
java.lang.Object
--java.awt.Component
--java.awt.Container
--javax.swing.JComponent
-- javax.swing.JProgressBar
```

当您在安装一个新软件时, 系统会告知您目前软件安装的进度如何, 才不会让您觉得程序好像死了。同样的, 若您设计的程序所需要的运行时间超过 2 秒以上, 您应该显示程序正在运行中的图标, 或直接显示程序运行的进度, 这样就能让用户清楚地知道程序到底是死了

还是继续在运行。在 Swing 中, JProgressBar 组件提供了类似这样的功能, 它可以很简单地输出进度的变化情况, 让您想要提供进度信息时, 不再需要自行绘制绘图组件, 只需要使用 JProgressBar 再加上几行程序设置就可以了。以下是 JProgressBar 的范例, 在此范例中, 我们使用 Timer 组件当作控制进度杆移动的速度, 当用户按下“Start”按钮时, 则进度杆线就会开始向右移动, 并显示出当前的进度信息。每当 JProgressBar 的值改变一次 (利用 setValue() 方法), 就会触发一次 ChangeEvent 事件, 如果您要处理这个事件, 就必须实现 ChangeListener 接口所定义的 stateChanged() 方法, 在此我们是将 JProgressBar 的移动信息放在了 label 上。

范例 ProgressBarDemo.java (文件位于随书光盘目录 exam\ch14\ProgressBar Demo.java)

```
1  import javax.swing.*;
2  import javax.swing.border.*;
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.event.*;
6
7  public class ProgressBarDemo implements ActionListener,ChangeListener
8  {
9      JFrame f = null;
10     JProgressBar progressbar;
11     JLabel label;
12     Timer timer;
13     JButton b;
14
15     public ProgressBarDemo()
16     {
17         f = new JFrame("progressbar Example");
18         Container contentPane = f.getContentPane();
19
20         label = new JLabel(" ",JLabel.CENTER);
21         progressbar = new JProgressBar();
22         progressbar.setOrientation(JProgressBar.HORIZONTAL);
23         progressbar.setMinimum(0);
24         progressbar.setMaximum(100);
25         progressbar.setValue(0);
26         progressbar.setStringPainted(true);
27         progressbar.addChangeListener(this);
28         progressbar.setPreferredSize(new Dimension(200,30));
29
30         JPanel panel = new JPanel();
31         b = new JButton("Start");
32         b.addActionListener(this);
33         panel.add(b);
34
35         timer = new Timer(50,this);
36
37         contentPane.add(panel,BorderLayout.NORTH);
38         contentPane.add(progressbar,BorderLayout.CENTER);
39         contentPane.add(label,BorderLayout.SOUTH);
40
41         f.pack();
```

```

42         f.setVisible(true);
43
44         f.addWindowListener(new WindowAdapter() {
45             public void windowClosing(WindowEvent e) {
46                 System.exit(0);
47             }
48         });
49     }
50
51     public static void main(String[] args)
52     {
53         new ProgressBarDemo();
54     }
55
56     public void actionPerformed(ActionEvent e)
57     {
58         if(e.getSource() == b)
59         {
60             timer.start();
61         }
62
63         if(e.getSource() == timer)
64         {
65             int value = progressbar.getValue();
66
67             if( value < 100)
68             {
69                 value++;
70                 progressbar.setValue(value);
71             }
72             else
73             {
74                 timer.stop();
75                 progressbar.setValue(0);
76             }
77         }
78     }
79
80     public void stateChanged(ChangeEvent e1)
81     {
82         int value = progressbar.getValue();
83
84         if(e1.getSource() == progressbar)
85         {
86             label.setText("目前已完成进度: "+Integer.toString(value)+" %");
87         }
88     }
89 }
90

```

说明:

- (1) 程序第 21 行, 建立一个水平的 JProgressBar 组件。
- (2) 程序第 22~25 行, 设置 JProgressBar 的方向、最小值、最大值与初始值。

- (3) 程序第 26 行, 设置是否在 `JProgressBar` 中显示出进度完成百分比的字符串。
- (4) 程序第 56~78 行, 处理按钮与 `Timer` 的 `ActionEvent` 事件。当按下按钮时, `Timer` 被激活, 此时 `progressbar` 会随着 `Timer delay` 的时间重新设置 `progressbar` 的进度值, 当进度值达到 100% 时, 停止 `Timer` 并设置 `progressbar` 的进度值为 0。
- (5) 程序第 80~88 行, 处理 `progressbar` 的 `ChangeEvent` 事件。每当 `progressbar` 的进度值改变时, 就会产生 `ChangeEvent` 事件, 在此利用 `getValue()` 方法取得 `progressbar` 的进度值, 并将此信息放在 `label` 上。

④ 程序运行结果如图 14-10 所示 (按下 “Start” 按钮后)。



图 14-10

若您将 `progressbar.setStringPainted(true)` 设置成 `false`, 则 `JProgressBar` 就不会输出进度信息, 如图 14-11 所示。



图 14-11

若您不喜欢 `JProgressBar` 中的显示文字, 您可以使用 `setString()` 方法加入自己想要显示的信息, 如 `progressbar.setString("Processing...")`; 运行结果如图 14-12 所示。



图 14-12

当然您也可以利用 `JComponent` 所提供的 `setBackground()` 与 `setForeground()` 方法来设置 `JProgressBar` 的背景与前景颜色, 如加上下面两行:

```
progressbar.setBackground(Color.yellow);  
progressbar.setForeground(Color.red);
```

◆ 则程序运行结果如图 14-13 所示。



图 14-13

若您不想显示 JProgressBar 的长条边框, 您可以使用 JProgressBar 的 `setBorderPainted()` 方法, 将它设置为 `false`, 如 “`progressbar.setBorderPainted(false);`”, 则运行结果如图 14-14 所示。

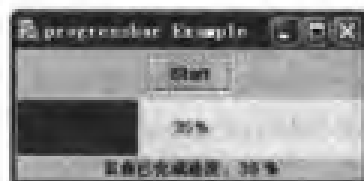


图 14-14

14-4 使用 Progress Monitor 组件

ProgressMonitor 的类层次结构图:

```
java.lang.Object
-- javax.swing.ProgressMonitor
```

当您将文件复制到另一个地方或在网络上下载文件时, 如果文件太大, 系统通常会弹出一个对话框告知您目前下载或复制的进度如何, 当下载或复制完成之后, 此对话框就会自动消失。要做到这样的功能, 您可以使用 JDialog 的传统方法, 然后利用一些 Threads 来控制动画的部分, 不过这个做法太过复杂。如果您只想要一个简单的进度对话框, 或是您不想编写太多复杂的程序代码, 您可以使用 Java 提供的 ProgressMonitor 组件。这个组件可以让您设置当系统运行时间超过多少秒钟时, 就会弹出一个对话框, 显示目前程序运行的进度。当运行进度达到 100% 时, 此对话框就会自动消失。表 14-3 是此 ProgressMonitor 的构造函数:

表 14-3

ProgressMonitor 构造函数

`ProgressMonitor(Component parentComponent, Object message, String note, int min, int max)`

建立一个 ProgressMonitor 对象, 所需设置的参数分别是母窗口、显示信息、显示说明与最大最小值。进度杆(Progress Bar)会从最小值开始跑, 跑到最大值后就自动关闭对话框

ProgressMonitor 组件利用 `setMillisToDecideToPopup()` 与 `setMillisToPopup()` 这两个方法来设置显示进度对话框的时机。`setMillisToDecideToPopup()` 是指当系统运行时间超过多少毫秒时, 就会出现进度对话框。而 `setMillisToPopup()` 是指当程序运行时间超过系统预估时间时, 就会出现进度对话框。例如某程序运行时间约 10 秒钟, 而我们将 `setMillisToDecideToPopup()` 方法设为 1 秒, 即 `setMillisToDecideToPopup(1000)`, 且将 `setMillisToPopup()` 方法设为 5 秒, 即

setMillisToPopup(5000)。当程序真正在运行时，由于系统预估运行时间（10 秒）大于 5 秒，因此会在程序运行一秒后出现进度对话框。不过读者要注意，这里所谓的系统预估时间并不十分准确，而且与个人的机器设备有相当大的关系。若您没有设置这两个方法的值，则 Java 会以默认值 setMillisToDecideToPopup(500)与 setMillisToPopup(2000)当作出现进度对话框的设置值。

在 ProgressMonitor 中利用 setProgress()方法设置进度杆(Progress Bar)“跑”的状态，您可以利用 setMinimum()与 setMaximum()方法来设置进度杆的最大最小值。当进度杆的值大于或等于最大值时，进度对话框会自动消失；或者是当用户按下进度对话框的“撤消”按钮时，对话框也会消失。您可以使用 ProgressMonitor 所提供的 isCanceled()方法来处理用户按下“撤消”按钮的事件，如中断文件下载等等。

ProgressMonitor 与 JProgressBar 较不同之处是 ProgressMonitor 在进度达到 100%后，对话框会自动消失且资源也将被系统回收，若要再产生对话框进度，就必须再从新建立一个 ProgressMonitor 对象。

在下面的范例中当用户按下“Start”按钮时，会连续输出 10 次图片。我们设置 setMillisToDecideToPopup()为 0, setMillisToPopup()为 1000, 而 Timer Delay 默认的时间为 500, 显示 10 张图中间需 9 次 Delay 时间，因此程序运行所需时间至少在 4500 毫秒以上，也就是大于 setMillisToPopup()所设的 1000 毫秒。所以此程序一运行时就会出现进度对话框：

范例 ProgressMonitorDemo.java (文件位于随书光盘目录 exam\ch14\ProgressMonitorDemo.java)

```
1  import javax.swing.*;
2  import javax.swing.border.*;
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.event.*;
6
7  public class ProgressMonitorDemo implements ActionListener,ChangeListener
8  {
9      JFrame f = null;
10     Timer timer = null;
11     ImageIcon[] icons = null;
12     JSlider slider = null;
13     JLabel label = null;
14     int index=0;    //设置要显示图片的索引值
15     int total=0;    //设置显示图次数的累加值
16     ProgressMonitor pMonitor = null;
17
18     public ProgressMonitorDemo()
19     {
20         f = new JFrame("Progress Monitor Example");
21         Container contentPane = f.getContentPane();
22
23         icons = new ImageIcon[5];
24         for (int i=0 ; i<5 ; i++)
25             icons[i] = new ImageIcon(".\\icons\\"+ (i+1)+".jpg");
26
27         label = new JLabel(icons[0]);
28
```

```

29         slider = new JSlider(JSlider.HORIZONTAL,
30                               0,    // min
31                               100,  //max
32                               50); //Initial value
33         slider.setPaintTicks(true);
34         slider.setMajorTickSpacing(20);
35         slider.setMinorTickSpacing(5);
36         slider.setPaintLabels(true);
37         slider.addChangeListener(this);
38
39         JPanel panel = new JPanel();
40         panel.setLayout(new GridLayout(1,2));
41
42         JButton b1 = new JButton("Start");
43         b1.addActionListener(new ButtonListener());
44         panel.add(b1);
45         JButton b2 = new JButton("Stop");
46         b2.addActionListener(new ButtonListener());
47         panel.add(b2);
48
49         panel.setPreferredSize(new Dimension(200,30));
50
51         timer = new Timer(slider.getValue()*10,this);
52
53         contentPane.add(slider,BorderLayout.NORTH);
54         contentPane.add(label,BorderLayout.CENTER);
55         contentPane.add(panel,BorderLayout.SOUTH);
56
57         f.pack();
58         f.setVisible(true);
59
60         f.addWindowListener(new WindowAdapter() {
61             public void windowClosing(WindowEvent e) {
62                 System.exit(0);
63             }
64         });
65     }
66
67     public static void main(String[] args)
68     {
69         new ProgressMonitorDemo();
70     }
71
72     public void actionPerformed(ActionEvent e)
73     {
74         if (pMonitor.isCanceled())
75         {
76             timer.stop();
77             index = 0;
78             total = 0;
79         } else {
80             pMonitor.setProgress(total*10);
81         }
82
83         if (total < 10)

```



```

84         {
85             if (index == 5)
86                 index = 0;
87             label.setIcon(Icons[index]);
88             label.repaint();
89             index++;
90             total++;
91         }
92     else
93         timer.stop();
94 }
95
96 public void stateChanged(ChangeEvent e1)
97 {
98     timer.setDelay(slider.getValue()*10);
99 }
100
101 class ButtonListener implements ActionListener
102 {
103     public void actionPerformed(ActionEvent e)
104     {
105         if (e.getActionCommand().equals("Start"))
106         {
107             pMonitor = new ProgressMonitor(f,
108                 "Showing Progress Monitor", //message
109                 "", //note
110                 0, //min
111                 100); //max
112
113             pMonitor.setNote("Changing Photo....");
114             pMonitor.setMillisToDecideToPopup(0);
115             pMonitor.setMillisToPopup(1000);
116             pMonitor.setProgress(0);
117             index = 0;
118             total = 0;
119             timer.start();
120         }
121         if (e.getActionCommand().equals("Stop"))
122         {
123             timer.stop();
124         }
125     }
126 }
127 }

```

⊕ 说明:

- (1) 程序第42与第46行,我们将处理按钮事件的程序代码写在 ButtonListener 这个 Inner Class 中。
- (2) 程序第 107~119 行,当用户按下“Start”按钮后,会产生一个 ProgressMonitor 组件,并设置其说明值 (setNote(“Changing Photo...”)) 与对话框出现时机 (setMillisToDecideToPopup(0)与 setMillisToPopup(1000))、设置进度对话框初始

值 (setProgress(0)等, 最后激活 Timer。

(3) 程序第 123 行, 当用户按下 “Stop” 按钮后, 停止 Timer。

(4) 程序第 72~94 行, 处理 Timer 所产生的(ActionEvent)事件。

(5) 程序第 74~81 行, 若用户按下进度对话框的 “撤消” 按钮时, 则停止 Timer 并将图片索引值与显示出图片的总数值回复为 0。否则显示出目前进度值 (setProgress(total*10))。

(6) 程序第 83~93 行, 当图片显示出次数超过 10 次时就会停止 Timer, 否则继续显示图。

◆ 程序运行结果如图 14-15 所示。

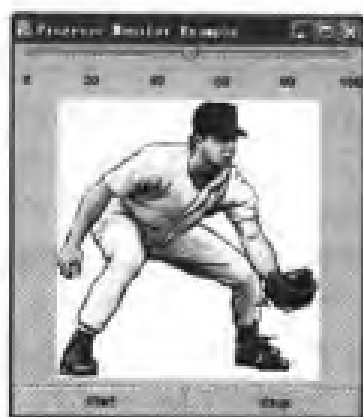


图 14-15

当按下 “Start” 按钮后, 如图 14-16 所示。



图 14-16

显示完 10 张图后此对话框会自动消失。若读者想更改此对话框的图案, 例如像 Windows 中删除文件的动画图标一般, 则必须利用 JDialog 组件, 不过这部分必须使用到 Thread, 有兴趣的读者可自行试试看。

14-5 使用 Progress Monitor Input Stream 组件

ProgressMonitorInputStream 的类层次结构图:

```
java.lang.Object
--java.io.InputStream
--java.io.FilterInputStream
-- javax.swing.ProgressMonitorInputStream
```

在这章的最后, 我们介绍一个比较特殊的进度(Progress)组件, 称为 ProgressMonitorInputStream, 它是 FilterInputStream 的子类。FilterInputStream 是继承 InputStream 抽象类而来,

主要用来处理程序读进来的数据流，用户可以利用这个类所提供的一些方法来过滤掉不想要的信息。

`ProgressMonitorInputStream` 这个组件主要也是用在读取数据的时候，可以监督读取数据的过程。例如当我们要把某一个文件读进来，而文件内容又很大时，我们可以利用 `ProgressMonitorInputStream` 显示出读文件的进度。或是读取一个相当大的数据库内容时，若在读取过程中没有显示适当的信息，用户很可能会误以为程序死掉了。

表 14-4 是 `ProgressMonitorInputStream` 的构造函数：

表 14-4

ProgressMonitorInputStream 构造函数
<code>ProgressMonitorInputStream(Component parentComponent, Object message, InputStream in)</code> 建立一个可监督读取 Input Stream 进度的组件(<code>ProgressMonitorInputStream</code>)

由上面的构造函数可以看出，`ProgressMonitorInputStream` 需要一个 `InputStream` 对象当作它的参数，若我们要监督读文件的过程，我们可以利用 `FileInputStream` 所得到的对象看成是 `InputStream` 的类型，来当作 `ProgressMonitorInputStream` 的参数。

`ProgressMonitorInputStream` 的使用方式跟 `ProgressMonitor` 组件一模一样，也就是说当读取数据时间过长时，系统会弹出一个进度对话框来告知用户目前已读取了多少文件内容。事实上，`ProgressMonitorInputStream` 本身并没有显示对话框的功能，而是利用其类所提供的 `getProgressMonitor()` 方法，来取得 `ProgressMonitor` 组件，并显示出进度对话框。在上面的 `ProgressMonitorInputStream` 构造函数中，读者可以发现并没有 `minimum` 与 `maximum` 的设置值，主要是因为系统会根据所读进来的数据流长度自动设置这两个参数值，因此用户无须自己设置。

使用 `ProgressMonitorInputStream` 所产生的进度对话框会有一个“撤消”按钮。当用户按下“撤消”按钮时，进度对话框会关闭，内部的读取数据操作也会中断，并丢出 `IOException`。

下面的范例中我们利用 `FileInputStream` 读取某个文件的内容，并将它显示在 `TextArea` 中，当程序在读取文件时，进度对话框会提示我们目前文件的读取状况。为避免读文件的操作过快，导致看不到进度对话框，我们利用 `Thread` 的 `sleep()` 方法来控制读文件的速度。

范例 `PMonitorInputStream.java` (文件位于随书光盘目录 `exam\ch14\PmonitorInputSteam.java`)

```
1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4  import java.io.*;
5
6  class PMonitorInputStream implements ActionListener
7  {
8      JFrame f = null;
9      JLabel label = null;
10     JTextArea textarea = null;
11     JFileChooser fileChooser = null;
12
13     public PMonitorInputStream()
```

```

14      {
15          f = new JFrame("ProgressMonitorInputStream Example");
16          Container contentPane = f.getContentPane();
17          textarea = new JTextArea();
18          JScrollPane scrollPane = new JScrollPane(textarea);
19          scrollPane.setPreferredSize(new Dimension(350,350));
20          JButton b = new JButton("读取文件");
21          b.addActionListener(this);
22          label = new JLabel(" ",JLabel.CENTER);
23
24          fileChooser = new JFileChooser();
25
26          contentPane.add(label,BorderLayout.NORTH);
27          contentPane.add(scrollPane,BorderLayout.CENTER);
28          contentPane.add(b,BorderLayout.SOUTH);
29
30          f.pack();
31          f.setVisible(true);
32
33          f.addWindowListener(new WindowAdapter() {
34              public void windowClosing(WindowEvent e) {
35                  System.exit(0);
36              }
37          });
38      }
39
40      public static void main(String[] args) {
41          new PMonitorInputStream();
42      }
43
44      public void actionPerformed(ActionEvent e)
45      {
46          File file = null;
47          int result = fileChooser.showOpenDialog(f);
48
49          textarea.setText("");
50
51          if (result == JFileChooser.APPROVE_OPTION)
52          {
53              file = fileChooser.getSelectedFile();
54              label.setText("您选择的文件名称为: "+file.getName());
55          }
56          else if(result == JFileChooser.CANCEL_OPTION)
57          {
58              label.setText("您没有选择任何文件");
59          }
60
61          FileInputStream inputStream;
62
63          if(file != null)
64          {
65              try{
66                  inputStream = new FileInputStream(file);
67              }catch(FileNotFoundException fe){
68                  label.setText("File Not Found");

```

```

69         return;
70     }
71
72     ProgressMonitorInputStream pmInputStream = new
73         ProgressMonitorInputStream(f, //parent component
74             "Get File Content.....", //message
75             inputStream); //input stream
76
77     ProgressMonitor pMonitor =
78     pmInputStream.getProgressMonitor();
79     pMonitor.setMillisToDecideToPopup(10);
80     pMonitor.setMillisToPopup(0);
81     int readbyte;
82
83     try{
84         while( (readbyte = pmInputStream.read()) != -1)
85         {
86             textarea.append(String.valueOf((char) readbyte));
87
88             try{
89                 Thread.sleep(10);
90             }catch(InterruptedException ie){}
91
92             if(pMonitor.isCanceled())
93             {
94                 textarea.append("\n\n 读取文件中断!! ");
95             }
96         }
97     }catch(IOException ioe){
98         label.setText("读取文件错误");
99     }
100     finally{
101         try{
102             if(pmInputStream != null)
103                 pmInputStream.close();
104             }catch(IOException ioe2){}
105         }
106     }
107 }
108 }

```

⊕ 说明:

- (1) 程序第 24 行, 我们利用 JFileChooser 组件让用户自行选择文件。
- (2) 程序第 44~107 行, 处理用户按下“读取文件”按钮的事件。
- (3) 程序第 47~56 行, 当用户选择完文件或按下“撤消”按钮后, JFileChooser 的 showOpenDialog() 方法会返回一个整数值, 判断用户是否选择了文件。若返回值为 JFileChooser.APPROVE_OPTION, 表示用户选择了文件, 若返回值为 JFileChooser.CANCEL_OPTION, 表示用户按下取消键。此部分在上一章中有详细的说明。

- (4) 程序第 72~75 行, 利用 `FileInputStream` 来产生 `ProgressMonitorInputStream` 对象, 并设置 `message` 的内容。
- (5) 程序第 77~80 行, 利用 `ProgressMonitorInputStream` 的 `getProgressMonitor()` 方法取得 `ProgressMonitor` 对象, 并设置进度对话框的显示时机。
- (6) 程序第 84~96 行, 读取 `InputStream` 的内容, 每次读一个 `byte`, 转成字符后显示在 `textarea` 上, 并利用 `Thread` 的 `sleep()` 方法减慢运行的速度。当 `read()` 方法读到 `InputStream` 结尾时, 会返回整数值 -1。
- (7) 程序第 100~105 行, 处理 `ProgressMonitorInputStream` 的关闭操作, 使系统回收相关资源。

⊕ 程序运行结果如图 14-17 所示 (按下“读取文件”按钮后)。

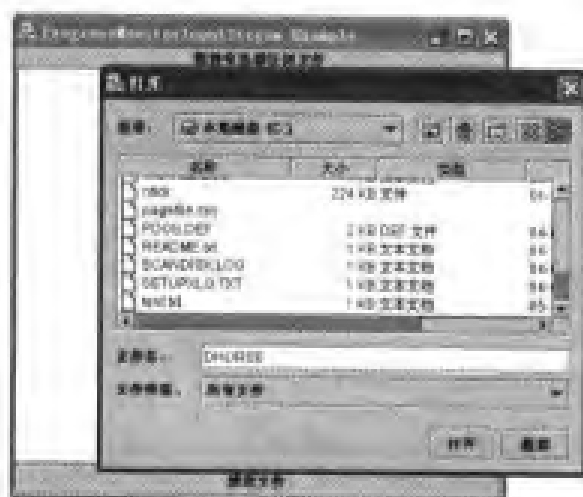


图 14-17

当您选择完一个文件后, 程序便开始运行读文件操作, 如图 14-18 所示。

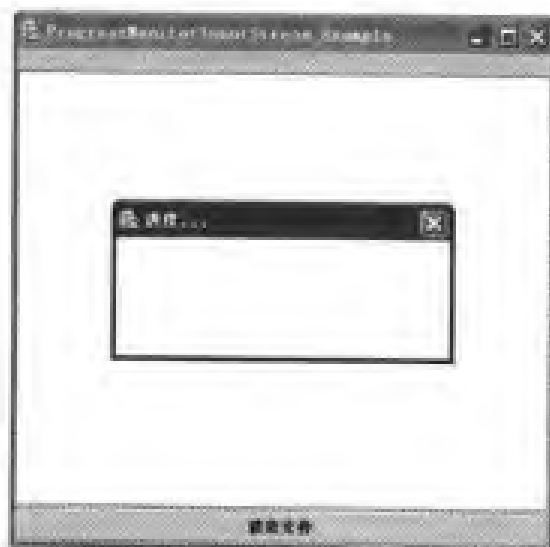


图 14-18

怎么进度对话框没显示任何东西呢？在这里有一点读者要特别注意，在 Swing 中有专门的运行线程（Thread）来处理事件的产生。因此在这个例子中，Thread 所需要处理的事件不仅仅是用户按下按钮，而且也需要处理 JProgressBar 中的变化情况与其他相关事件（读者还记得改变 JProgressBar 的变化情况会产生 ChangeEvent 吗？），若此时还要处理 ProgressMonitor InputStream 的读文件程序，便会产生 Thread 无法同时处理太多事情，导致画面无法实时显示的问题（称为 Race Condition，如上例所示）。要解决这个问题，我们必须产生另一个新的 Thread 独立处理读取文件内容的部分，这样整个画面的显示就不会有问题了，以下是我们修改后的程序：

范例 PMInputStreamFixed.java (文件位于随书光盘目录 examch14\PMInputStreamFixed.java)

```

1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4  import java.io.*;
5
6  class PMInputStreamFixed implements ActionListener, Runnable
7  {
8      JFrame f = null;
9      JLabel label = null;
10     JTextArea textarea = null;
11     JFileChooser fileChooser = null;
12
13     public PMInputStreamFixed()
14     {
15         f = new JFrame("ProgressMonitorInputStream Example");
16         Container contentPane = f.getContentPane();
17         textarea = new JTextArea();
18         JScrollPane scrollPane = new JScrollPane(textarea);
19         scrollPane.setPreferredSize(new Dimension(350, 350));
20         JButton b = new JButton("读取文件");
21         b.addActionListener(this);
22         label = new JLabel(" ", JLabel.CENTER);
23
24         fileChooser = new JFileChooser();
25
26         contentPane.add(label, BorderLayout.NORTH);
27         contentPane.add(scrollPane, BorderLayout.CENTER);
28         contentPane.add(b, BorderLayout.SOUTH);
29
30         f.pack();
31         f.setVisible(true);
32
33         f.addWindowListener(new WindowAdapter() {
34             public void windowClosing(WindowEvent e) {
35                 System.exit(0);
36             }
37         });
38     }
39
40     public static void main(String[] args) {

```

```

41         new PMInputStreamFixed();
42     }
43
44     Thread athread;
45
46     public void actionPerformed(ActionEvent e)
47     {
48         athread = new Thread(this);
49         athread.start();
50     }
51
52     public void run()
53     {
54         File file = null;
55         int result = fileChooser.showOpenDialog(f);
56
57         textarea.setText("");
58
59         if (result == JFileChooser.APPROVE_OPTION)
60         {
61             file = fileChooser.getSelectedFile();
62             label.setText("您选择的文件名称为: "+file.getName());
63         }
64         else if (result == JFileChooser.CANCEL_OPTION)
65         {
66             label.setText("您没有选择任何文件");
67         }
68
69         FileInputStream inputStream;
70
71         if (file != null)
72         {
73             try{
74                 inputStream = new FileInputStream(file);
75             } catch (FileNotFoundException fe){
76                 label.setText("File Not Found");
77                 return;
78             }
79
80             ProgressMonitorInputStream pmInputStream = new
81                 ProgressMonitorInputStream(f,    //parent component
82                     "Get File Content.....", //message
83                     inputStream);    //input stream
84
85             ProgressMonitor pMonitor =
86 pmInputStream.getProgressMonitor();
87             pMonitor.setMillisToDecideToPopup(10);
88             pMonitor.setMillisToPopup(0);
89             int readbyte;
90
91             try{
92                 while( (readbyte = pmInputStream.read()) != -1)
93                 {
94                     textarea.append(String.valueOf((char) readbyte));
95

```



```

96         try{
97             Thread.sleep(10);
98         }catch(InterruptedException ie){}
99
100         if(pMonitor.isCanceled())
101         {
102             textArea.append("\n\n 读取文件中断!! ");
103         }
104     }
105 }catch(IOException ioe){
106     label.setText("读取文件错误");
107 }
108 finally{
109     try{
110         if(pmInputStream != null)
111             pmInputStream.close();
112     }catch(IOException ioe2){}
113 }
114 }
115 }
116 }

```

⊕ 说明:

- (1) 程序第 6 行, 我们利用实作 Runnable 接口来实现 Thread 所要做的事情。
- (2) 程序第 46~50 行, 当用户按下“读取文件”按钮时就会产生一个新的 Thread, 专门用来处理读取文件数据流的事情。使用 Thread 的 start() 方法就会激活 Thread, 并运行 run() 方法中的程序代码。

⊕ 程序运行结果如图 14-19 所示。



图 14-19

您可以发现此时的进度对话框与 TextArea 的文字显示相当的顺畅, 不再有刚才无法实时显示的问题产生。

14-6 本章总结

在本章中，我们完整地介绍了与 Progress Bar 相关的组件。包括 JSlider：可以让用户拖曳滑动杆来微调程序的进行，例如控制 Timer 的 delay 时间。Timer：可以控制一种周期性的变化事件，如更换图片或定时更新数据（这对看股市信息的人可说是相当重要）。JProgressBar：可以显示目前的进度状况，让用户清楚知道系统的运行信息。ProgressMonitor：跟 JProgressBar 相似，不过它是出现另一个对话框来显示进度状况，并可设置显示时机，它的内部还是使用 JProgressBar 来显示进度杆的信息。最后的 ProgressMonitorInputStream：与 ProgressMonitor 相似，主要是用于显示目前读取数据的进度，并以输入流（InputStream）为参数，若您的系统需要经常处理大量的数据流，这个组件就相当的好用。

14-7 本章习题

1. 试分别建立水平与垂直方向的滑动杆，并标记刻度与数字。
2. 试以 Timer 组件设计出一个会跑的文字动画，或是更改 TimerDemo1.java 程序，使它成为具有较完整功能的显示图程序。
3. 试说明如何控制 ProgressMonitor 中进度对话框的出现时机，并说明 ProgressMonitor、ProgressMonitorInputStream 与 JProgressBar 的不同处。
4. 试以 JDialog 组件搭配 Thread 功能作出如 JProgressBar 的功能，并以图 14-20 表示。



图 14-20

5. 试修改 PMInputStreamFixed.java 程序，使得数据输入来源是数据库而不是文件。

15

创造用户最熟悉的环境 (Look and Feel)

创造用户最熟悉的环境——Look and Feel，什么是 Look and Feel 呢？简单地说，就是用户看到的环境即自己最熟悉的环境。例如某用户是 Windows 操作系统的使用者，当您要介绍某套软件给他使用时，软件的操作界面却以 Unix 环境来表现，相信他一定使用得非常不顺手，可能因此而对该软件的印象打了一个大大的折扣。因此在本章中我们将告诉您如何制作出不同操作平台的环境界面，让您的软件界面可因不同的使用环境而表现出相对应的效果。

深入淺出

Chapter 15: Look and Feel (Look and Feel) (Look and Feel) (Look and Feel) (Look and Feel) (Look and Feel)

15-1 为什么要用 Look and Feel

当我们在设计一套系统或一个好用的程序时，良好的人机界面（UI，User Interface）是非常重要的。一个简单易于使用的人机界面能让用户快速的掌握该系统或程序的组织模式和操作方法，让用户能专注于系统或程序的使用或开发上，而不会在工作之前还需要花很长的时间来了解人机界面的操作方式。一个最典型的例子就是命令行模式（Command Mode）（如 MS-DOS、UNIX 的 SHELL）和窗口模式（Window Mode）（如 Window 98、X-window）的差别。在命令行模式下，若是我们不事先知道各个指令所代表的意义，便无法去使用这些系统的资源，这就是指令行模式在人机界面设计上的缺点之一。相对的，在窗口模式下，我们很容易就知道该怎样复制文件、删除文件……，这是因为窗口模式的人机界面采用较容易了解的可视化（Visualize）设计，让用户很快就能熟悉窗口的操作方法。因此，如何规划出容易使用与易于了解的人机界面是程序设计者的重大责任。

因此，我们在开发 Java 应用程序时也应当规划程序的人机界面。由于 Java 是一种具有跨平台特性的程序语言，能在各种不同的操作平台上运行同一个程序，所以设计出该平台的特定环境在 Java 中更显得特别重要。但从前面章节的范例中我们可以发现，不论我们的程序在哪个操作平台上运行，所展现出来的人机界面都不会有什么改变：这是因为 Java 在人机界面的规划方式上有一套标准的机制，虽然是在不同的操作系统平台上运行同一个程序，但是通过 Java 虚拟机（JVM）的解译后，会调用到相同的标准类库（Library），因此我们会在不同操作系统上看到相同的软件画面。这样的做法的确能让 Java 的用户有一种标准的感觉，但却没有考虑到原本就已经习惯特定操作平台的用户。如一位在 X-window 下操作 10 年的用户，平常只用计算机处理一些帐单工作，对计算机本身并不是很熟悉，若今天他接触一套由 Java 所设计的帐务管理软件，整个软件的环境为 Java 的默认环境，则他必须抛开以前 X-Window 的感觉从新接受新的用户界面，这可能需要花他一些时间来适应新的环境，也可能让他拒绝使用新的软件。这些都不是软件公司所愿意见到的。为了让用户不会再遇到这样的麻烦，Java 的设计者在规划 Swing 时就加入了 Look and Feel 这一部分来解决这个问题。通过 Look and Feel 的机制我们可以很简单地改变 Java 程序里人机界面的类型，来配合各种不同操作平台的模式，创造用户最熟悉的作业环境。

◆ 说明：

Look and Feel 就是本章的主题，为了让读者能更了解这个名词的含义，因此我们用“创造用户最熟悉的环境”来作为标题。由于一般我们是使用 Java 的默认环境来作为我们开发系统的工具（如前面章节大部分的范例），并没有更改用户界面的显示方法，而这个默认环境我们就称为 Java Look and Feel。

在这章之前，由于我们已经学习了所有 Swing 组件的使用方法，因此从本章开始，我们将慢慢把一些 Swing 组件集合起来一起使用，让读者对 Swing 组件有更深刻的印象。

15-2 什么是 Look and Feel

Look and Feel 就是提供程序设计者任意转换程序的人机界面，来对应到不同的操作平台

中。这也就是说我们只要利用 Look and Feel 的机制就可以设计出适合我们所熟悉的工作环境，而不再只是一成不变的单一的界面。Look and Feel 最大的优点在于展现不同窗口特色的人机界面，我们先来看看各种不同操作系统的人机界面，如图 15-1 所示。图中最底层的窗口是 UNIX 系列的窗口模式 (CDE Style)，第二层为麦金塔系列的窗口模式 (Macintosh Style)，最上层的则是微软系列的窗口模式 (Microsoft Windows Style)。另外，当然还有 Java Style 的窗口模式，不论你在任何操作系统平台上工作，只要这个平台支持 JFC，那么 Java Style 的窗口模式将会是在 Java 运行环境下的默认值。



图 15-1

由底层往外分别是 CDE Style、Macintosh Style、Microsoft Windows Style、Java Style (Java Look and Feel)，如图 15-2 所示。

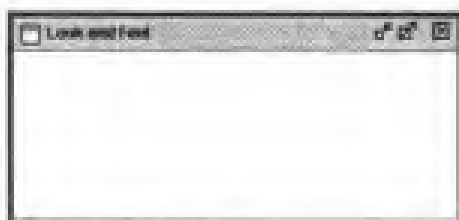


图 15-2

在看过了各个系统不同的人机界面样式后，是不是发现原来界面也可以有这么多的变化呢。然而如何来变换这些界面？该怎么利用 Look and Feel 技巧来编写程序？我们来看下面各小节介绍。

◆ 说明：

在 JDK1.1 和 Java2 SDK 中支持的 Look and Feel 如下：

- (1) Java Look and Feel，一般在程序中我们叫做 Metal Look and Feel。不论我们在任何一个操作系统上工作，只要这个操作系统支持 JFC，Java Look and Feel 就是默认的 Look and Feel。
- (2) Microsoft Windows，一般在程序中我们叫做 Windows Look and Feel。这个 Look and Feel 只能在 Microsoft Windows 的操作系统 (Windows 95/98/NT/2000) 下使用。
- (3) CDE，一般在程序中我们叫做 CDE/Motif Look and Feel。这个 Look and Feel 是用

在 UNIX 系列的操作系统。它是仿真在 Solaris* 操作系统 2.6 版中的 OSF/Motif 1.2.5 版。（注：Solaris 为 Sun 公司出品的操作系统）

- (4) Macintosh style Look and Feel, 一般在程序中我们叫做 Mac OS Look and Feel, 这个 Look and Feel 只能在 Macintosh 的操作系统下使用。

15-3 在 Java 中如何使用 Look and Feel

我们在上一节中提到过 Java Look and Feel 是 JFC 默认的 Look and Feel, 也就是说当我们在编写 Java 程序时, 如果不改变 Look and Feel 的设置值, 那么当程序在运行时我们所看到的界面就是 Java Style 的界面。我们先来看一个范例, 在这个范例中我们一开始使用 Java Look and Feel 的设置值, 若要更改环境界面, 可利用菜单所提供的功能来作变换:

15-3-1 Look and Feel 范例一

在这个范例中我们是在一个 JFrame 中加入了 JMenuBar、JLabel、JButton、JCheckBox、JList 组件, 在 JMenuBar 中我们提供一个 JMenu, 让用户选择所想要的 Look and Feel 环境。

范例 LF1.java (文件位于随书光盘目录 exam\ch15\LF1.java)

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class LF1 implements ActionListener
6  {
7      JFrame f = null;
8      JButton b = null;
9      JMenuItem item1 = null;
10     JMenuItem item2 = null;
11     JMenuItem item3 = null;
12
13     LF1() {
14         f = new JFrame("Look and Feel");
15         Container contentPane = f.getContentPane();
16         contentPane.setLayout(new GridLayout(1,3));
17
18         JMenuBar menubar = new JMenuBar();
19         JMenu menu = new JMenu("选择想要的 Look_and_Feel");
20         item1 = new JMenuItem("JAVA Look and Feel");
21         item2 = new JMenuItem("Windows Look and Feel");
22         item3 = new JMenuItem("Motif Look and Feel");
23         item1.addActionListener(this);
24         item2.addActionListener(this);
25         item3.addActionListener(this);
26         menu.add(item1);
27         menu.add(item2);
28         menu.add(item3);
29         menubar.add(menu);
30         f.setJMenuBar(menubar);

```

```

31
32     JPanel p1 = new JPanel();
33     p1.setLayout(new GridLayout(2,1));
34     JLabel label = new JLabel("下面是按钮");
35     JButton b = new JButton("按我",new ImageIcon(".\\icons\\hand.jpg"));
36     b.setHorizontalTextPosition(JButton.CENTER);
37     b.setVerticalTextPosition(JButton.BOTTOM);
38     b.addActionListener(this);
39     p1.add(label);
40     p1.add(b);
41
42     JPanel p2 = new JPanel();
43     p2.setLayout(new GridLayout(3,1));
44     p2.setBorder(BorderFactory.createTitledBorder("快餐店"));
45     JCheckBox c1 = new JCheckBox("麦当劳");
46     JCheckBox c2 = new JCheckBox("肯德基");
47     JCheckBox c3 = new JCheckBox("21 世纪");
48     p2.add(c1);
49     p2.add(c2);
50     p2.add(c3);
51
52     String[] s = {"10","12","14","16","18"};
53     JList list = new JList(s);
54     list.setBorder(BorderFactory.createTitledBorder("文字大小"));
55
56     contentPane.add(p1);
57     contentPane.add(p2);
58     contentPane.add(list);
59
60     f.pack();
61     f.show();
62     f.addWindowListener(new WindowAdapter() {
63         public void windowClosing(WindowEvent e) {
64             System.exit(0);
65         }
66     });
67 }
68
69 public static void main(String[] args)
70 {
71     new LF1();
72 }
73
74 public void actionPerformed(ActionEvent e)
75 {
76     if(e.getSource() == b)
77     {
78         JFrame newf = new JFrame("新窗口");
79         newf.setSize(200,200);
80         newf.show();
81     }
82     if(e.getSource() == item1)
83     {
84         try {
85             UIManager.setLookAndFeel("

```



```
86     javax.swing.plaf.metal.MetalLookAndFeel");
87         SwingUtilities.updateComponentTreeUI(f);
88         f.pack();
89     }
90     catch( Exception e1 ) {
91         System.out.println ("Look and Feel Exception");
92         System.exit(0);
93     }
94 }
95 if(e.getSource() == item2)
96 {
97     try {
98         UIManager.setLookAndFeel("
99 com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
100        SwingUtilities.updateComponentTreeUI(f);
101        f.pack();
102    }
103    catch( Exception e1 ) {
104        System.out.println ("Look and Feel Exception");
105        System.exit(0);
106    }
107 }
108 if(e.getSource() == item3)
109 {
110     try {
111         UIManager.setLookAndFeel("
112 com.sun.java.swing.plaf.motif.MotifLookAndFeel");
113        SwingUtilities.updateComponentTreeUI(f);
114        f.pack();
115    }
116    catch( Exception e1 ) {
117        System.out.println ("Look and Feel Exception");
118        System.exit(0);
119    }
120 }
121 }
122 }
```

◆ 说明:

- (1) 这个程序使用到的组件与事件处理方法,在这之前的章节都已经详细介绍过了,因此就不再多述。
- (2) 这个程序所要提的重点是在第 85~88 行,当用户选择了某种“Look and Feel”后,系统必须显示出用户所选择的界面。如第 85~86 行,我们要设置某种 Look and Feel,必须使用 UIManager 类所提供的 setLookAndFeel()静态方法,然后在 setLookAndFeel()方法中传入 Look and Feel 的类参数。例如若要显示 Java 的环境界面则传入“javax.swing.plaf.metal.MetalLookAndFeel”类名称,若要显示 Windows 的环境界面则传入“com.sun.java.swing.plaf.windows.WindowsLookAndFeel”类名称,若要显示 Unix 的环境界面则传入“com.sun.java.swing.plaf.motif.MotifLookAndFeel”类名称。

- (3) 程序第 87 行, 当我们开始运行程序后, 想要改变原有的 Look and Feel 面貌, 并不需要重新激活程序, 只要利用 SwingUtilities 类所提供的 `updateComponentTreeUI()` 静态方法, 就可以变换不同的环境界面。
- (4) 程序第 88 行, 由于每一个环境界面所显示的组件大小都不太相同, 因此利用 `pack()` 方法, 可使组件的大小更为美观与适当。
- (5) 程序第 98~101、第 111~114 行的意思同上。

⊕ 程序运行结果如下:

一开始激活时由于我们未设置 Look and Feel 环境, 因此默认值为 Java Look and Feel, 如图 15-3 所示。



图 15-3

当用户拉下菜单选择 “Windows Look and Feel” 时, 将如图 15-4 所示。



图 15-4

当用户拉下菜单选择 “Motif Look and Feel” 时, 将如图 15-5 所示。



图 15-5

不知读者有没有发现, Motif Look and Feel 所显示的大小比较大一点点, 而且外观明显不同于其他两个。另外我们也发现虽然我们选取了不同环境的 Look and Feel, 但对于 `JFrame f` 而言, 整体的外观并没有改变, 例如 `JFrame f` 最上面的标题栏不管是使用哪种 Look and Feel, 最终所显示的一律是蓝色渐层标题栏加上 Windows 系列的最小化、最大化与关闭按钮。这是

为什么呢？原因就是我们在第3章所介绍的，JFrame 与 JApplet 都不是 lightweight 组件，而是 heavyweight 组件，因此 JFrame 与 JApplet 均是 OS dependent（Operating System dependent），当使用的操作系统不同时，所显示出来的 JFrame 或 JApplet 就会有所不同。由于笔者所使用的操作系统是 Windows XP，因此所显示的 JFrame 自然就显示 Windows 的外观了。

在上面的例子中，我们利用 UIManager 类所提供的 setLookAndFeel() 静态方法（static method）传入 Look and Feel 的类参数；若我们不晓得 Java 所默认的 Look and Feel 类名称呢？我们可以利用 UIManager 类所提供的 getCrossPlatformLookAndFeelClassName() 静态方法，就可以取得 Java 默认的 Look and Feel 类名称“javax.swing.plaf.metal.MetalLookAndFeel”。或是利用 getSystemLookAndFeelClassName() 静态方法，取得当前操作平台的 Look and Feel 类名称。由于笔者所使用的操作系统是 Windows 操作系统，因此我们可以将上个范例的 74~122 行改写成：

```

74         public void actionPerformed(ActionEvent e)
75         {
76             if(e.getSource() == b)
77             {
78                 JFrame newf = new JFrame("新窗口");
79                 newf.setSize(200,200);
80                 newf.show();
81             }
82             if(e.getSource() == item1)
83             {
84                 try {
85                     UIManager.setLookAndFeel(
86                         UIManager.getCrossPlatformLookAndFeelClassName());
87                     SwingUtilities.updateComponentTreeUI(f);
88                     f.pack();
89                 } catch( Exception e1 ) {
90                     System.out.println ("Look and Feel Exception");
91                     System.exit(0);
92                 }
93             }
94             if(e.getSource() == item2)
95             {
96                 try {
97                     UIManager.setLookAndFeel(
98                         UIManager.getSystemLookAndFeelClassName());
99                     SwingUtilities.updateComponentTreeUI(f);
100                    f.pack();
101                }
102                catch( Exception e1 ) {
103                    System.out.println ("Look and Feel Exception");
104                    System.exit(0);
105                }
106            }
107            if(e.getSource() == item3)
108            {
109                try {
110                    UIManager.setLookAndFeel("
111                    com.sun.java.swing.plaf.motif.MotifLookAndFeel");
112                    SwingUtilities.updateComponentTreeUI(f);

```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100

```

113         f.pack();
114     }
115     catch( Exception e1 ) {
116         System.out.println ("Look and Feel Exception");
117         System.exit(0);
118     }
119 }
120 }
121 }

```

程序运行结果当然与上例一模一样。

接下来我们介绍一个比较复杂的例子，让读者更能感受到各种 **Look and Feel** 的不同之处。

15-3-2 Look and Feel 范例二

在这个范例中，我们在 `JFrame` 里增加了目录菜单项目和工具栏，另外还有一个展示树状结构的对象，并利用到 `JInternalFrame` 的功能。

范例 LookAndFeel.java(文件位于随书光盘目录文件 exam\ch15\LookAndFeel.java)

```

1  import javax.swing.*;
2
3  public class LookAndFeel{
4
5      public static void main(String[] args){
6
7          JFrame F = new JFrame();
8          F.setVisible(true);
9      }
10 }

```

⊕ 说明:

为了将程序模块化，我们将程序的各个部分拆成不同的类来编写。这个程序只是一个开头，重要的是我们在第 7 行中新建了一个名为 `MainFrame` 的 `JFrame` 对象，至于这个对象的内容请看下面的程序代码：

范例 MainFrame.java (文件位于随书光盘目录文件 exam\ch15\MainFrame.java)

```

1 import javax.swing.*;
2 import java.awt.event.*;
3 import java.awt.*;
4
5 public class MainFrame extends JFrame{
6
7     JDesktopPane desktop;
8     JMenuBar      MBar;
9     JToolBar      toolBar;
10
11     public MainFrame(){
12
13         super("MainFrame-Look and Feel");
14         setBounds(100,100,600,400);

```

```
15     buildContent();
16     buildMenu();
17     buildToolBar();
18
19     this.getContentPane().add(toolBar, BorderLayout.NORTH);
20
21     this.addWindowListener(new WindowAdapter() {
22     public void windowClosing(WindowEvent e) {
23         quit();
24     }
25     });
26 } //end of main
27
28 protected void buildContent() {
29     desktop = new JDesktopPane();
30     getContentPane().add(desktop);
31 } //end of buildContent()
32
33 protected void buildMenu(){
34     MBar = new JMenuBar();
35     MBar.setOpaque(true);
36     JMenu mfile = buildFileMenu();
37     JMenu mdemo = buildDemoMenu();
38
39     mfile.setMnemonic('F');
40     mdemo.setMnemonic('D');
41
42     MBar.add(mfile);
43     MBar.add(mdemo);
44     setJMenuBar(MBar);
45 } //end of bulidMenu()
46
47 protected void buildToolBar(){
48     toolBar = new JToolBar();
49     toolBar.setFloatable(true);
50
51     ToolbarAction tba_new =
52         new ToolbarAction("new", new ImageIcon("icons/new24.gif"));
53     ToolbarAction tba_open =
54         new ToolbarAction("open", new ImageIcon("icons/open24.gif"));
55     ToolbarAction tba_close =
56         new ToolbarAction("close", new ImageIcon("icons/copy24.gif"));
57     ToolbarAction tba_save =
58         new ToolbarAction("save", new ImageIcon("icons/save24.gif"));
59
60     JButton JB;
61     JB = toolBar.add(tba_new);
62     JB.setActionCommand("TB_NEW");
63     JB.setToolTipText((String)tba_new.getValue(Action.NAME));
64     JB = toolBar.add(tba_open);
65     JB.setActionCommand("TB_OPEN");
66     JB.setToolTipText((String)tba_open.getValue(Action.NAME));
67     JB = toolBar.add(tba_close);
68     JB.setActionCommand("TB_CLOSE");
69     JB.setToolTipText((String)tba_close.getValue(Action.NAME));
```

```

70
71     toolBar.addSeparator();
72
73     JB = toolBar.add(tba_save);
74     JB.setActionCommand("TB_SAVE");
75     JB.setToolTipText((String)tba_save.getValue(Action.NAME));
76     tba_save.setEnabled(false);
77 }//end of buildToolBar()
78
79 public void quit(){
80     System.exit(0);
81 }//end of quit()
82
83 public JMenu buildFileMenu() {
84
85     JMenu file = new JMenu("File");
86     JMenuItem newf = new JMenuItem("New");
87     JMenuItem open = new JMenuItem("Open");
88     JMenuItem close= new JMenuItem("Close");
89     JMenuItem quit = new JMenuItem("Exit");
90
91     newf.setEnabled(false);
92     open.setEnabled(false);
93     close.setEnabled(false);
94
95
96     newf.setMnemonic('N');
97     open.setMnemonic('O');
98     close.setMnemonic('C');
99     quit.setMnemonic('X');
100
101     newf.setAccelerator( KeyStroke.getKeyStroke
102 ('N',java.awt.Event.CTRL_MASK,false) );
103     open.setAccelerator( KeyStroke.getKeyStroke
104 ('O',java.awt.Event.CTRL_MASK,false) );
105     close.setAccelerator( KeyStroke.getKeyStroke
106 ('C',java.awt.Event.CTRL_MASK,false) );
107     quit.setAccelerator( KeyStroke.getKeyStroke
108 ('X',java.awt.Event.CTRL_MASK,false) );
109
110     quit.addActionListener(new ActionListener() {
111         public void actionPerformed(ActionEvent e) {
112             quit();
113         }
114     });
115     file.add(newf);
116     file.add(open);
117     file.add(close);
118     file.addSeparator();
119     file.add(quit);
120     return file;
121 }//end of buildFileMenu()
122
123 protected JMenu buildDemoMenu() {
124     JMenu demo = new JMenu("Demo");

```

```

125     JMenuItem tree = new JMenuItem("Tree Structure");
126
127     tree.setMnemonic('T');
128     tree.setAccelerator( KeyStroke.getKeyStroke
129 ('T', java.awt.Event.CTRL_MASK, false) );
130
131     tree.addActionListener(new ActionListener() {
132         public void actionPerformed(ActionEvent e) {
133             DemoTree();
134         }
135     });
136     demo.add(tree);
137     return demo;
138 } //end of buildDemoMenu()
139
140 public void DemoTree() {
141     JInternalFrame JItree = new DemoTree();
142     desktop.add(JItree, new Integer(1));
143     try {
144         JItree.setVisible(true);
145         JItree.setSelected(true);
146     } catch (java.beans.PropertyVetoException e2) {}
147 } //end of DemoTree()
148
149 } //end of class MainFrame

```

⊕ 说明:

- (1) 由于 MainFrame 是一个 JFrame 对象, 所以这个对象必须继承 JFrame 对象, 如第 5 行中所示。我们在这个对象中除了设置窗口的对象, 还加入了目录菜单和工具栏, 另外还有一个展示树状结构的对象。
- (2) 在第 13 行和第 14 行中, 分别设置了这个窗口 (MainFrame) 的标题和大小。在第 29 行中, 我们新建了一个 JDesktopPane 对象, 并在第 30 行中将 JDesktopPane 的对象加入窗口 (MainFrame) 中。
- (3) 在第 33 行~45 行的区段中, 我们建立了一个目录菜单, 并将这个菜单加入到窗口中。在其中第 35 行的 setOpaque(true) method 则是将 JMenuBar 的不透明的属性设为 true, 使得 JMenuBar 的显示能自动重绘。第 36、37 行则是建立菜单中的 File 和 Demo 选项。第 39、40 行则是设置选项的快捷键 (使用【Alt+快捷键】就可运行), 第 42、43 行是将选项对象加入目录菜单中。建立 File 子菜单的区段在第 83~120 行, 其中的第 101~108 行是为各个子菜单的选项设置快捷键 (使用【Ctrl+快捷键】就可运行)。而建立 Demo 子菜单的区段在第 122~137 行, 其中的第 133 行调用了 DemoTree 这个类, 这个类在第 139~147 行, 我们可以在第 141 行中看到这个类属于一个 JInternalFrame 的类, 关于这个类的内容我们在说明后面会有详细地介绍。
- (4) 在第 47~77 行的区段中, 我们建立了一个工具栏, 并在第 19 行中将这个工具栏加入到窗口中。在第 49 行我们将工具栏的浮动值设为 true, 使得工具栏能够被移

动到窗口的各个位置。在第 51 到 58 行利用 `ToolBarAction` 类分别将各个图标加载到工具栏上。第 60 到 76 行分别设置各个图标的对应操作, 在这里我们专注于人机界面的设计上, 所以这部分就留给有兴趣的读者们再进一步地去钻研。

范例 DemoTree.java (文件位于随书光盘目录文件 examch15\ DemoTree.java)

```

1  import javax.swing.*;
2  import javax.swing.tree.*;
3
4  public class DemoTree extends JInternalFrame{
5
6      public DemoTree(){
7          super("Demo Tree Structure", true, true, true, true);
8
9          DefaultMutableTreeNode manager;
10         DefaultMutableTreeNode leader;
11         DefaultMutableTreeNode engineer;
12
13         DefaultMutableTreeNode top =
14         new DefaultMutableTreeNode("Empolyee List");
15
16         top.add( manager = new DefaultMutableTreeNode("Manager") );
17         top.add( leader  = new DefaultMutableTreeNode("Leader") );
18         top.add( engineer = new DefaultMutableTreeNode("Engineer") );
19
20         manager.add( new DefaultMutableTreeNode("C. Fan") );
21         manager.add( new DefaultMutableTreeNode("C. Tomas") );
22         manager.add( new DefaultMutableTreeNode("C. Simth") );
23
24         leader.add( new DefaultMutableTreeNode("K. Jacky") );
25         leader.add( new DefaultMutableTreeNode("M. Shu") );
26
27         engineer.add( new DefaultMutableTreeNode("E. Kevin") );
28         engineer.add( new DefaultMutableTreeNode("H. Alex") );
29         engineer.add( new DefaultMutableTreeNode("G. J.") );
30         engineer.add( new DefaultMutableTreeNode("L. Kate") );
31         engineer.add( new DefaultMutableTreeNode("F. Mike") );
32
33         JTree tree = new JTree(top);
34         JScrollPane treeScroller = new JScrollPane(tree);
35         treeScroller.setBackground(tree.getBackground());
36         setContentPane(treeScroller);
37         setSize( 250, 200);
38         setLocation( 200, 20);
39     } //end of DemoTree()
40 } //end of class DemoTree

```

⊕ 说明:

- (1) 这个程序是继承了 `JInternalFrame`。由于在程序中会用到 `Tree` 的类, 因此我们在第 2 行要将 `javax.swing.tree` 的 package 包含进来。

- (2) 在建立树状结构的节点时，我们会用到 `DefaultMutableTreeNode` 类。而各个节点间的从属关系是由 `add()` 方法来决定。
- (3) 第 13、14 行建立的节点为根节点。第 16~18 行建立第一层子节点并通过 `add()` 方法来链接根节点和第一层节点。第 20~22 行、第 24~25 行、第 27~31 行分别建立第二层节点（即各第一层子节点下的节点）。当我们组织完 `tree` 的结构后，在第 33 行将根节点指定到 `JTree` 的类中。另外我们在显示树状结构时是一个下拉式的显示方式，因此我们需要使用 `JScrollPane` 类来放置 `JTree` 类，使得树状数据在显示时能完整的显示。

在编译并运行 `Java LookAndFeel` 并点选选项之后的结果如图 15-6 所示。（只捕获重点部分）



图 15-6

在这里我们可以看到组件以 `Java Look and Feel` 所显示出来的样子。那么我们该怎样将这个程序的界面转换成其他不同样式的界面呢？其实很简单，除了 `LookAndFeel` 这个类外，其余所有的类和对象都不需要做任何改动。我们马上来看看如何来显示 `Microsoft Windows` 的程序界面。我们将修改过的 `LookAndFeel.java` 存成 `winLookAndFeel.java`，马上来看看这段程序的内容：

范例 `winLookAndFeel.java` (文件位于随书光盘目录文件 `exam\ch15\winLookAndFeel.java`)

```
1  import javax.swing.*;
2
3  public class winLookAndFeel{
4
5      public static void main(String[] args){
6
7          try {
8              UIManager.setLookAndFeel(
9                  "com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
10             }
11             catch ( UnsupportedLookAndFeelException e ) {
12                 System.out.println ("This Look & Feel not supported on this
13 platform. \nProgram Terminated");
14                 System.exit(0);
15             }
16             catch ( IllegalAccessException e ) {
17                 System.out.println ("This Look & Feel could not be accessed.
```


范例 CDELookAndFeel.java (文件位于随书光盘目录文件 exam\ch15\CDELookAndFeel.java)

```

1  import javax.swing.*;
2
3  public class winLookAndFeel{
4
5      public static void main(String[] args){
6
7          try {
8              UIManager.setLookAndFeel("
9              com.sun.java.swing.plaf.motif.MotifLookAndFeel");
10         }
11         catch ( UnsupportedOperationException e ) {
12             System.out.println ("This Look & Feel not supported on this platform.
13             \nProgram Terminated");
14             System.exit(0);
15         }
16         catch ( IllegalAccessException e ) {
17             System.out.println ("This Look & Feel could not be accessed.
18             \nProgram Terminated");
19             System.exit(0);
20         }
21         catch ( ClassNotFoundException e ) {
22             System.out.println ("This Look & Feel could not found.
23             \nProgram Terminated");
24             System.exit(0);
25         }
26         catch ( InstantiationException e ) {
27             System.out.println ("This Look & Feel could not be instantiated.
28             \nProgram Terminated");
29             System.exit(0);
30         }
31         catch ( Exception e ) {
32             System.out.println ("Unexpected error. \nProgram Terminated");
33             e.printStackTrace();
34             System.exit(0);
35         }
36
37         JFrame F = new MainFrame();
38         F.setVisible(true);
39     } // end of main
40 } //end of class winLookAndFeel

```

⊕ 说明:

这段程序和原本的 LookAndFeel.java 比较起来只差在第 7~35 行。其中最重要的就是在第 8、9 两行, 在这里我们将 UIManager 的 LookAndFeel 设置成 com.sun.java.swing.plaf.motif.MotifLookAndFeel。其余的部分为使用这个方法的例外处理部分, 共有: 此操作平台不支持的 Look and Feel、无法被存取的 Look and Feel、找不到 Look and Feel 类、Look and Feel 类安装错误等可能发生的情况。

在编译并运行 Java CDELookAndFeel 并点选选项之后的结果如图 15-8 所示。(只捕获重

点部分)



图 15-8

虽然在 JFC 默认的 Look and Feel 是 Java Look and Feel, 不过 Java Look and Feel 也有其对应的设置方式, 如下面这个范例所示:

范例 javaLookAndFeel.java (文件位于随书光盘目录文件 exam\ch15\ javaLookAndFeel.java)

```

1  import javax.swing.*;
2
3  public class winLookAndFeel{
4
5      public static void main(String[] args){
6
7          try {
8              UIManager.
9              setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
10             {
11                 catch ( UnsupportedOperationException e ) {
12                     System.out.println ("This Look & Feel not supported on this platform.
13                     \nProgram Terminated");
14                     System.exit(0);
15                 }
16                 catch ( IllegalAccessException e ) {
17                     System.out.println ("This Look & Feel could not be accessed.
18                     \nProgram Terminated");
19                     System.exit(0);
20                 }
21                 catch ( ClassNotFoundException e ) {
22                     System.out.println ("This Look & Feel could not found.
23                     \nProgram Terminated");
24                     System.exit(0);
25                 }
26                 catch { InstantiationException e } {
27                     System.out.println ("This Look & Feel could not be instantiated.
28                     \nProgram Terminated");
29                     System.exit(0);
30                 }
31                 catch { Exception e } {
32                     System.out.println ("Unexpected error. \nProgram Terminated");

```

```
33     e.printStackTrace();
34     System.exit(0);
35 }
36
37 JFrame F = new MainFrame();
38 F.setVisible(true);
39 } // end of main
40 } //end of class WinLookAndFeel
```

⊕ 说明:

这个程序和原本的 LookAndFeel.java 运行后的效果是一模一样的。一般来说,若是我们要使用 Java Look and Feel 时是不会加上程序中第 7~35 行的部分,但是加上这个区段后可以使程序较容易理解与维护。另外,若有使用到切换不同 Look and Feel 的功能时,我们也应该了解如何处理 Java Look and Feel 的设置方式。

15-4 本章总结

在本章中,我们说明了为什么要使用 Look and Feel、什么是 Look and Feel、各种不同操作系统特色的窗口模式及如何在 Java 中使用 Look and Feel。活用 Look and Feel 会让用户感受到程序界面的亲和度,除此之外也能让已经习惯于特定操作平台的用户节省习惯新的窗口环境所需要的时间。因此适时的使用 Look and Feel 优点来创造出用户最熟悉的环境是身为一个 UI 设计师所必须拥有的理念,让我们为 Java 程序界面注入更丰富的想象力吧,共勉之!

15-5 本章习题

1. JDK1.1 和 Java 2 支持的 Look and Feel 有哪些呢?
2. JFC 默认的 Look and Feel 是哪一种呢?
3. 当我们在 Windows 操作系统下使用 JDK 时,可以使用 Macintosh Style 的 Look and Feel 吗?
4. 请在 MainFrame.java 中,将工具栏上的相关功能实现。
5. 试写一支 Swing 程序,请分别比较 Java、Windows 与 Unix Look and Deel 的不同处。



16

整合范例

在了解了大部分的 Swing 组件后，让我们再从头到尾地复习一次 Swing 的各种组件和其应用方式。在这里我们将编写一个简单的编辑器（Editor）让读者知道各个组件的使用方式与作用，并了解组件间在整合应用时可能遇到的各种问题。希望读者在看完本章后能对 Swing 有一个更完整的认识。



16-1 建立窗口

窗口 (Frame) 是所有可视化程序设计的基础组件, 因此我们以构造窗口作为介绍本程序的开端。下面这个范例是构造一个基础窗口的框架, 在这个程序中只将桌面面板 (Desktop Pane) 放到基础窗口中:

范例 LookAndFeel.java (文件位于随书光盘目录 exam\ch16\step01\LookAndFeel.java)

```
1  import javax.swing.*;
2
3  public class LookAndFeel{
4
5      public static void main(String[] args){
6
7          JFrame F = new MainFrame();
8          F.setVisible(true);
9      } // end of main
10 } //end of class LookAndFeel
```

⊕ 说明:

为了将程序模块化, 我们将程序的各个部分拆成不同的类来编写。这段程序只是一个开头, 重要的是我们在第 7 行中新建了一个名为 MainFrame 的 JFrame 对象, 至于这个对象的内容请看下面的程序代码。

范例 MainFrame.java (文件位于随书光盘目录 exam\ch16\step01\MainFrame.java)

```
1  import javax.swing.*;
2  import java.awt.event.*;
3
4  public class MainFrame extends JFrame{
5
6      JDesktopPane desktop;
7
8      public MainFrame(){
9
10         super("MainFrame");
11         setBounds(100,100,400,200);
12         buildContent();
13
14         this.addWindowListener(new WindowAdapter() {
15             public void windowClosing(WindowEvent e) {
16                 quit();
17             }
18         });
19     } //end of MainFrame()
20
21     protected void buildContent() {
22         desktop = new JDesktopPane();
23         getContentPane().add(desktop);
```

```
24     //end of buildContent()  
25  
26     public void quit(){  
27         System.exit(0);  
28     } //end of quit()  
29 } //end of class MainFrame
```

⊕ 说明:

- (1) 程序第 4 行, 由于 MainFrame 继承了 JFrame 类, 因此具有 JFrame 的性质。
- (2) 程序第 10~11 行, 分别设置了这个窗口 (MainFrame) 的标题和大小。
- (3) 程序第 22 行, 我们新建了一个 JDesktopPane 对象, 并在第 23 行中将 JDesktopPane 的对象加入窗口 (MainFrame) 中。

编写好之后, 我们运行编译操作:

```
javac LookAndFeel.java
```

编译完成后, 再运行 java LookAndFeel。

我们可以看到以下的运行结果, 如图 16-1 所示。



图 16-1

我们可以看到这个基础窗口包含了标题栏及标题 (左上角文字) 和窗口控制组件 (右上角三个按钮), 并且能够自由的改变窗口的大小。

16-2 菜单与工具栏 (Menus and Toolbars)

16-2-1 菜单 (Menus)

当我们创造出一个窗口后, 接下来就是要把这个窗口所需要的控制选项放到适当的位置上。提到控制窗口的选项, 我们第一个会想到的就是菜单 (Menu) 上的各个选项, 如文件 (File)、编辑 (Edit)、说明 (Help) 等和其分类下的各个分项功能, 如文件下的新建文件 (New)、打开文件 (Open) 及离开 (Exit) 等。那么我们现在就马上来看一个例子, 说明如何建立我们所需要的菜单。

◆ 说明:

接下来的每一个范例都是延续前一个范例而来,也就是一直延伸 MainFrame.java 的程序代码,并加入更多的协力对象。我们在编译程序时,依旧只要编译 LookAndFeel.java 即可。也就是说在运行程序时,只要键入 java LookAndFeel 就能够开始运行。

范例 MainFrame.java(文件位于随书光盘目录 exam\ch16\step02\MainFrame.java)

```

1  import javax.swing.*;
2  import java.awt.event.*;
3
4  public class MainFrame extends JFrame{
5
6      JDesktopPane desktop;
7      JMenuBar      MBar;
8
9      public MainFrame(){
10
11          super("MainFrame-Look and Feel");
12          setBounds(100,100,600,400);
13          buildContent();
14          buildMenu();
15
16          this.addWindowListener(new WindowAdapter() {
17              public void windowClosing(WindowEvent e) {
18                  quit();
19              }
20          }); //end of addWindowListener
21      } //end of main
22
23      protected void buildContent() {
24          desktop = new JDesktopPane();
25          getContentPane().add(desktop);
26      } //end of buildContent()
27
28      protected void buildMenu(){
29          MBar = new JMenuBar();
30          MBar.setOpaque(true);
31          JMenu mfile = buildFileMenu();
32
33          MBar.add(mfile);
34
35          setJMenuBar(MBar);
36      } //end of bulidMenu()
37
38      public void quit(){
39          System.exit(0);
40      } //end of quit()
41
42      public JMenu buildFileMenu() {
43
44          JMenu file = new JMenu("File");
45          JMenuItem newf = new JMenuItem("New");

```

```
46      JMenuItem open = new JMenuItem("Open");
47      JMenuItem close= new JMenuItem("Close");
48      JMenuItem quit = new JMenuItem("Exit");
49
50      file.add(newf);
51      file.add(open);
52      file.add(close);
53      file.addSeparator();
54      file.add(quit);
55      return file;
56  } //end of buildFileMenu()
57  } //end of class MainFrame
```

◆ 说明:

- (1) 这段程序是在展示如何将菜单加入界面中。程序第 14 行，我们在 MainFrame 的构造函数中调用 buildMenu()方法来建立菜单，而所调用到的 buildMenu()方法在程序的第 28~36 行。
- (2) 在 buildMenu()方法里，我们先在第 29 行中新建一个 JMenuBar 的对象来准备存放各菜单的数据。第 30 行的 setOpaque(true) method 则是将 JMenuBar 的不透明的属性设为 true，使得 JMenuBar 的显示能自动重绘。
- (3) 程序第 31 行，调用 buildFileMenu()方法来建立 FileMenu 的菜单数据。buildFileMenu()方法在程序第 42~56 行。在第 44 行中，我们建立了一个 JMenuItem 的对象来存放 File 菜单的内容，第 45~48 行的 JMenuItem 对象就是各项菜单的数据。
- (4) 程序第 50~56 行在第 50~54 行中，我们把数据(JMenuItem)加入 File 菜单(JMenu)中，其中第 53 行为分隔线(Separator)。在第 53 行中将 File 菜单(JMenu)加到 JMenuBar 对象中。另外最重要的是在第 35 行，将 JMenuBar 对象加到 JFrame 上，让 MenuBar 能显现出来。

在编译并运行了 java LookAndFeel 之后的结果如图 16-2 和图 16-3 所示。



图 16-2

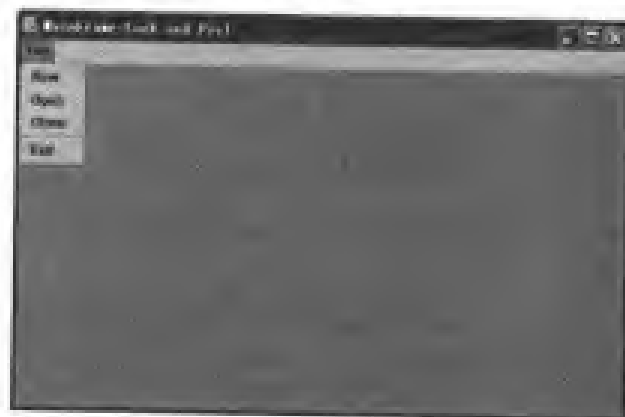


图 16-3

到这里我们已经把菜单成功的加入界面中了，不过我们还未在菜单中加入快捷键的功能，并且也没有相应的运行效果。下面我们再次修改 MainFrame.java 程序，使它具有上述的效果。

范例 MainFrame.java (文件位于随书光盘目录 exam\ch16\step03\MainFrame.java)

```

1  import javax.swing.*;
2  import java.awt.event.*;
3
4  public class MainFrame extends JFrame{
5
6      JDesktopPane desktop;
7      JMenuBar      MBar;
8
9      public MainFrame(){
10
11          super("MainFrame-Look and Feel");
12          setBounds(100,100,600,400);
13          buildContent();
14          buildMenu();
15
16          this.addWindowListener(new WindowAdapter() {
17              public void windowClosing(WindowEvent e) {
18                  quit();
19              }
20          }); //end of addWindowListener
21      } //end of main
22
23      protected void buildContent() {
24          desktop = new JDesktopPane();
25          getContentPane().add(desktop);
26          } //end of buildContent()
27
28      protected void buildMenu(){
29          MBar = new JMenuBar();
30          MBar.setOpaque(true);
31          JMenu mfile = buildFileMenu();
32
33          mfile.setMnemonic('F');
34
35          MBar.add(mfile);
36          setJMenuBar(MBar);
37          } //end of bulidMenu()
38
39      public void quit(){
40          System.exit(0);
41          } //end of quit()
42
43      public JMenu buildFileMenu() {
44
45          JMenu file = new JMenu("File");
46          JMenuItem newf = new JMenuItem("New");
47          JMenuItem open = new JMenuItem("Open");
48          JMenuItem close= new JMenuItem("Close");
49          JMenuItem quit = new JMenuItem("Exit");
50
51          close.setEnabled(false);
52
53          newf.setMnemonic('N');
54          open.setMnemonic('O');
55          close.setMnemonic('C');
```

```

56         quit.setMnemonic('X');
57
58         newf.setAccelerator( KeyStroke.getKeyStroke
59 ('N',java.awt.Event.CTRL_MASK,false) );
60         open.setAccelerator( KeyStroke.getKeyStroke
61 ('O',java.awt.Event.CTRL_MASK,false) );
62         close.setAccelerator( KeyStroke.getKeyStroke
63 ('C',java.awt.Event.CTRL_MASK,false) );
64         quit.setAccelerator( KeyStroke.getKeyStroke
65 ('X',java.awt.Event.CTRL_MASK,false) );
66
67         newf.addActionListener(new ActionListener() {
68             public void actionPerformed(ActionEvent e) {
69                 makeNewFrame();
70             }
71         });
72
73         open.addActionListener(new ActionListener() {
74             public void actionPerformed(ActionEvent e) {
75                 openDocument();
76             }
77         });
78
79         quit.addActionListener(new ActionListener() {
80             public void actionPerformed(ActionEvent e) {
81                 quit();
82             }
83         });
84
85         file.add(newf);
86         file.add(open);
87         file.add(close);
88         file.addSeparator();
89         file.add(quit);
90         return file;
91     } //end of buildFileMenu()
92
93     public void makeNewFrame() {
94         JInternalFrame JInew = new NewFrame();
95         desktop.add(JInew, new Integer(1));
96         try {
97             JInew.setVisible(true);
98             JInew.setSelected(true);
99         } catch (java.beans.PropertyVetoException e2) {}
100     } //end of makeNewFrame
101
102     public void openDocument() {
103         JFileChooser chooser = new JFileChooser();
104         chooser.showOpenDialog(this);
105     } //end of openFrame()
106 } //end of class MainFrame

```

◆ 说明:

- (1) 首先我们在第 33 行利用 `setMnemonic()` 这个方法加入 File 的快捷键, 并指定为按键 **【F】**, 运行程序后只要按 **【Alt+F】** 即可下拉 File 菜单。另外在第 53~56 行中, 我们也是使用 `setMnemonic()` 方法设置快捷键, 也就是在相关字母上加上下划线, 用户只要按下划线所标示的字母就能运行相关的操作了。程序第 58~65 行, 设置